EECS150: Spring 2010 Project
Computer System with Accelerated Graphics

UC Berkeley College of Engineering
Department of Electrical Engineering and Computer Science

Revision A

1 Time Table

<table>
<thead>
<tr>
<th>ASSIGNED</th>
<th>Friday, February 26th</th>
</tr>
</thead>
<tbody>
<tr>
<td>DUE</td>
<td>Week 16: May 4th, during your assigned lab section</td>
</tr>
</tbody>
</table>

2 Motivation

The primary goal of this project is to familiarize EECS150 students with modern methods and tools used in digital design.

In order to make the project both interesting and useful, we will guide you through the design and implementation of a miniature, yet complete, digital system. This semester, the digital system you will be designing is a simple 3-stage CPU implementing a subset of the MIPS ISA. In addition to this, you will implement a portion of a very real DVI video interface and a simple graphics accelerator capable of rendering colored lines. Figure 1 shows an abstract view of how the components in this system are brought together to form a system. Getting a functional implementation running is and will be your primary goal. To better expose you to real design decisions and tradeoffs, however, we are requiring that you optimize your design for area (FPGA resource utilization) while attaining a specified minimum clock frequency.

![MIPS150 High-Level Architecture](image)

Figure 1: MIPS150 High-Level Architecture

You will use Verilog HDL to implement this system, targeting a Xilinx XUPv5 platform (powered by a Virtex-5 LX110T FPGA). You will use industry-standard CAD tools including Mentor Graphics’
ModelSim, Synopsis’ SynplifyPro, and the Xilinx ISE toolflow to map your Verilog to actual resources on the FPGA. The project will give you experience designing RTL, resolving hazards in a simple pipeline, building interfaces, and teach you how to approach system-level optimization.

In tackling these challenges, your first step will be to map our high level specification to a design which can be translated into a hardware implementation. After that, you will produce and debug that implementation. These first steps can potentially take significant time if you have not thought out your design prior to trying implementation. After you have built a working implementation, the next step will be optimizing it for area (cost, resource use) on the target FPGA. You will be expected to produce a relatively minimal circuit, implementing the required functionality, given a clock fixed at a certain frequency. At the end of this second phase (optimization, post implementation), you will have a greater understanding for the development process of digital hardware.

As in previous semesters, your EECS150 project is possibly the largest project you have faced so far. Good time management and good design organization will help you succeed in this course.

2.1 Philosophy

This document is meant to describe a high-level specification for the project and its associated support hardware. You can also use it to help lay out a plan for completing the project. As with any design you will encounter in the professional world, we are merely providing a framework within which your project must fit. (Unlike other college classes we will not tell you exactly what to write or how to design your project.)

You should consider the TAs a source of direction and clarification, but it is up to you to produce a fully functional design targeting the XUPv5 boards. We will attempt to help, when possible, but ultimately the burden of designing and debugging your solution lies on you.

In the end, what counts is having a well-implemented, bug free and optimized project by Week 16: May 4th, during your assigned lab section. Due on Week 16: May 4th, during your assigned lab section, is everything described and required in the project specifications that you will be getting in the coming weeks. This semester, you are able to receive some extra credit by completing your project ahead of schedule. This is optional and is described in detail in Section 4.5. There is also the opportunity to extend our framework with additional functionality for extra credit. This more traditional extra credit is described in Section 4.6.

2.2 General Project Tips

Make sure to use top-down design methodologies in this project. We began by taking the problem of designing a basic computer system, modularizing it into distinct parts, and then refining those parts into manageable checkpoints. You should take this scheme one step further; we have given you each checkpoint, so break each into smaller and manageable pieces. If you follow this guideline, and our interface specifications, you should be able to split the project up between you and your partner and for a more efficient partnership.

As with many engineering disciplines, digital design has a normal development cycle. In the norm, after modularizing your design, your strategy should roughly resemble the following steps:

**Design** your modules well, make sure you understand what you want before you begin to code.

**Code** exactly what you designed; do not try to add features without redesigning.

**Simulate** thoroughly; writing a good testbench is as much a part of creating a module as actually coding it.

**Debug** completely; anything which can go wrong with your implementation will.

**Verify** your implementation on live hardware frequently.

**Version** your project (only the source code that you wrote!) via SVN frequently, with descriptive commit messages.
Document your project thoroughly, as you go. Your design review documents will help, but you should never forget to comment your Verilog and to keep your diagrams up to date. Aside from the final project report (you will need to turn in a report documenting your project), you can use your design documents to help the debugging process. Finish the required features first. Attempt extra features after everything works well. If your submitted project does not work by the final deadline, you will not get any credit for any extra credit features you have implemented.

3 Project Description

The project, as has been done in past semesters, will be divided into checkpoints. This section will summarize the content of the checkpoint documents released with each project checkpoint. It is the complete, high-level project specification for each checkpoint. For more detail please refer to the appropriate checkpoint or lab lecture.

3.1 SMIPS150 Processor

The processor consists of a simple 3-stage pipeline, implementing only a small subset of instructions from the MIPS ISA. It features memory-mapped I/O to allow it to communicate with other components in your project. With that in mind, the processor is the heart of your project. It serves both as a controller for and a bridge between the various components that make up the project. Refer back to Figure 1 for a top level picture of how the processor fits in with the rest of your project.

Even though we will be putting a few restrictions on what you can and cannot do for your processor, the implementation details will be largely left for you to decide. Your design is expected to be clean, efficient, and optimized for area, while still being able to meet all of the imposed requirements.

3.2 Communication Interfaces

Once you finish designing your pipelined processor, you will need a means of interacting with it through your desktop or laptop computer. This is accomplished through the serial interface, which you have (largely) implemented in Lab 5, as well as a high-speed Ethernet interface that you will use to transport larger amounts of data in bulk.

3.2.1 Serial Interface

From the top level, the serial interface is composed of your laptop/desktop’s terminal program (PUTTY or HyperTerminal), the UART inside of your desktop/laptop, the serial cable, the UART on the FPGA, and the adaptor which you designed in Lab 5.

Having the serial interface working, will allow you to communicate with a program running on the MIPS150 processor. We will supply a default “boot monitor” program with a small set of commands. You will include the binary for this program in your design and it will be loaded into your instruction memory as part of the FPGA configuration process. That way, when the processor is reset it begins executing the boot monitor program. With it you will be able to “peek”\(^1\) and “poke”\(^2\) memory locations, load data and instructions at into your processor’s instruction and data memories at high-speed using the Ethernet connection, and otherwise control what your processor is doing. The boot monitor also functions somewhat like a BIOS program on a PC, by supplying special functions to user programs (such as sending and receiving characters over the serial port).

3.2.2 Ethernet Interface

Even though the serial interface provides a shell-like interface to your project, you will quickly find that it offers too little bandwidth to truly enable your project. For this reason, you will extend your system with a high-speed Ethernet interface, allowing for fast file transfer. To implement the hardware interface, you will interface with the dedicated Ethernet MAC (EMAC) primitive block on your FPGA.

---

\(^1\)To peek at state is to read its value without disrupting it.

\(^2\)To poke at state is to explicitly alter it.
The EECS150 staff will provide you with a software TFTP\(^3\) client implementation for your processor. As you will learn, TFTP is composed of both a client and server. The TFTP server will run on your laptop or desktop PC using standard TFTP software that the staff will provide.

After implementing the Ethernet interface, you will be able to quickly download files to your system and upload files from your system. Although TFTP is considered a “file” transfer protocol, files themselves can be anything: programs, data, video, images, etc. Thus, this component enables high-speed communication to and from your project in general.

3.3 Frame Buffer

The frame buffer is essentially a large dual-ported memory dedicated to store pixel data, organized into a rectangular array corresponding to the screen resolution. The contents of this memory are periodically read out into the video stream and sent to the video output device (TV/monitor). In other words, the frame buffer stores an uncompressed bitmap of the image to be shown on the screen. The CPU (and later the graphics accelerator) will write to the frame buffer to update pixels on the screen.

In this case, the frame buffer will store a 800x600 image writable via a memory-mapped interface, and readable by a DVI video interface. The video interface expects each pixel to be formatted as a 24-bit RGB triplet (8 bits for each of red, green, and blue color channels). Doing the math, it is not hard to see that the SRAM on the XUPv5 does not have enough on-chip memory to accommodate the entire frame in full color. For this reason, we will reduce the color depth somewhat, and encode the RGB triplet as a 16-bit halfword in the 5R6G5B format (5 bits for the red component, 6 for green, and 5 for blue). The green component is given a greater range, as the human eye is most sensitive to this color, making good use of the added precision.

An off-Chip 256Kx36 SRAM will be used to store pixel data. We will pack 2 pixels per 36-bit word in memory (leaving some of the SRAM unused). In the implementation of the Frame Buffer, you will learn to communicate with this off-chip device, and implement a rather typical memory interface. The SRAM is single-ported, so you will have to figure out how to arbitrate memory accesses to provide the dual-ported framebuffer interface. A DVI video interface will be provided for you to interface the framebuffer with the DVI chip that sends signals to the video output device.

3.4 Vector Graphics Accelerator

Although the CPU can render any image via direct per-pixel access to the frame buffer, you will quickly find that the processor draws lines very slowly. The reason for this is the processor’s inefficiency is the need to serialize all computation. By analyzing the parallelism present in a given algorithm, we can construct a very fast circuit - a hardware accelerator - to run that algorithm.

Each group’s project is required to have the ability to accelerate the drawing of lines. To accomplish this task, you will take advantage of an algorithm from the discrete digital analyzer (DDA) family known as Bresenham’s line algorithm. The accelerator you implement will allow lines to be quickly output into the frame buffer. In addition to the speed benefit of having an autonomous graphics engine, the CPU is also freed to perform other useful tasks while a line is being drawn.

3.5 Optimizations

In addition to maintaining a working implementation by the final deadline, you will also be responsible for optimizing your design for area given a certain clock frequency. You will have to be able to run your design at a certain clock frequency. If you cannot (such as the case if you don’t finish all of the required functional blocks), you will not receive extra credit for other features that you have implemented (see Section 5 for details). After meeting the specified frequency, you will optimize your implementation in order to produce the most area-efficient (fewest FPGA resources) circuit possible. Credit for your optimizations will be calculated using a cost function. At a high level, the cost function will look like:

\[
\text{Cost} = C_{\text{LUT}} \times \#\text{of LUTs} + C_{\text{RAMB}} \times \#\text{of RAMBs} + \ldots
\]

\(^{3}\text{Trivial File Transfer Protocol.}\)
where $C_{LUT}$ and $C_{RAMB}$ are constant value weights that will be decided upon based on how much each resource that you use should cost.

In the coming weeks we will provide a program use to compute the cost of your design. We will use the spread of costs across the class in order to determine grades. Keep in mind that cost is only one component of your project grade. Correct functionality is far more important.

# 4 Project Administration

This section discusses the policies associated with each part of the project. It also discusses what will be expected of you by the end of the project and throughout the entire semester. In overview, you will have weekly design reviews with your TA(s). When a checkpoint is due, you will be required demonstrate it (have it “checked off”) during your lab section. At each checkpoint, you will be graded on functionality, quality of design, and style (good design documents, clean Verilog, etc.). At the end of the project, you will be required to present your project in its entirety (“final checkoff”). A short semi-formal project report will be required at the conclusion of the semester.

## 4.1 Design Reviews

For each checkpoint, you are to meet with your TA(s) for a design review. As you improve your design, additional design reviews can be administered based on TA availability. **Both partners must be present at each design review.** In very unusual circumstances, you may schedule an appointment for a design review outside of lab hours.

You will receive a grade at the design review, according to the following scale:

2 points for a design that illustrates your understanding of the problem and your effort in solving it.

1 point will be given for limited understanding of the problem statement, or lacking documentation of your design (almost all designs we have seen for lab 5 fall under this category).

0 points will be given if your design is either non-existent or is severely lacking in either content or presentation.

Design review grades are part of your total project grade.

The design document should illustrate your understanding of the problem and your attempt at the solution. **Do not bring Verilog to your design review!** Verilog captures your implementation in gory detail, rather than your design at a high level, so it is inappropriate for a design review.

Instead, show high-level diagrams and visual aides. Block diagrams and bubble-arc diagrams are excellent for this purpose. Waveform diagrams are useful in some rare cases where per-cycle timing must be shown for an interface or a small circuit. In general, each diagram must have a label, briefly placing it in context of the task at hand.

We are not looking for any specific items in your designs. Instead, we are looking for an in-depth understanding of the problem statement, and a thought-out attempt at a solution. The point of design reviews is for the staff to give you feedback on your design, and to get you thinking clearly about the hardware you will be building before you actually implement the circuit.

## 4.2 Checkpoints

We have divided the project into checkpoints so that you (and the staff) can pace your progress. Each checkpoint will be due in your lab section during a specified week. Refer to the calendar page on the website or to Section 6 for the project timeline.

**In-lab Checkoff Procedure**

You will first be required to demo your progress to your TA(s) in your designated lab section. This demo will allow us to quickly assess your progress. We will ask some questions about your project, and require
you to run a set of tests to show functional correctness. We will then assign a Functionality grade as follows:

2 points will be given for an implementation that appears to correctly implement all required features.

1 point will be given for an implementation that is “mostly there,” or exhibiting a known and understood bug. Some cases of non-deterministic behavior also fall under this section.

0 points will be given for a severely flawed or incomplete implementation.

Finally, we will assign a “Style” grade based on your design documentation and Verilog.

2 points will be given for a well-organized, well-documented design.

1 point will be given for a design with some organization and documentation.

0 points will be given for a mess of illegible Verilog or severely misleading documentation.

More information on the “Style” component is given in Section 4.3.

4.3 Style Grade & Project Report

Your “style” grade will be composed of several sections:

1. The quality of your Verilog and documentation when checked off for each checkpoint (see Section 4.2). If your Verilog is incomprehensible, or deviates significantly from the CS150 guidelines, you may lose significant credit.

2. Your final project report (see below).

Upon completing the project, you will be required to submit a report detailing the progress of your CS150 project. The report should document your final circuit at a high level, and describe the design process that led you to your implementation. We expect you to document and justify any significant tradeoffs you have made throughout the semester, as well as any pitfalls and lessons learned. Additionally, you will document any optimizations made to your system, evaluate your project in terms of area, and other characteristics that set your project apart others. The report, including diagrams, should be on the order of ten pages long.

This report will not be difficult if you maintain good design documentation throughout the course of the project. As a guideline, think of the report as a supporting document for a 5 minute discussion about your CS150 experience at an interview, etc. The report is your medal for completing CS150, and is often indeed a good document to show at an interview. A more detailed description will be made available closer to the due date.

4.4 Final Checkoff

Final checkoff is the final presentation of your project to the teaching staff.

All files required to build your project (run through synthesis, translate, map and PAR) must be submitted before your allotted time for checkoff. This applies to all files you wish the grader to consider (all files needed to reconstruct your project from source). No material you submit after the final checkoff date will be considered in computing your grade.

More details regarding final checkoff will be made in the coming weeks.

4.5 Early Checkoff

The project may be turned in and checked off one week early for extra credit. These submissions will be graded by the same standard as other submissions. Submitting your project early means that you may not submit during the regular deadline. As such, you have less time to work on your project.

To find which day early checkoff is held on, see Section 6. For information on how much credit is given for early checkoff, see Section 5.

These tests will not be exhaustive, and will not guarantee correctness of your implementation. Write your own tests for verification.
4.6 Additional Functionality

If you finish the required functionality early, you may consider adding additional features to your design. In general, this type of addendum must be intellectually interesting, present design challenges, and be implemented using the practices we have taught you in the course.

If you have an interesting idea, present it to your TA in the form of a design (similar to a design review). You must come prepared with a design documented, having read the literature required to implement what it is you want to build, in order to qualify for extra credit in this category. In accordance with this policy, points will be allocated on a case by case basis. No extra features will receive credit unless approved by the TAs well ahead of time.

For information on how much credit is given for additional functionality, see Section 5.

4.7 Late Policy

If you are late in making any deadline you will receive zero credit for that deadline. This policy applies to final checkoff, design reviews, and regular checkpoints. Exceptions to this policy are conflicts brought up and approved by the TAs ahead of time or documented illness.

5 Project Grading

Based on what was discussed in Section 4, project grades will be assigned as shown below. Note that the project (as a whole) constitutes 35% of your course grade.

60% Functionality at the final checkoff. Your design will be subjected to a comprehensive test suite and your score will reflect the functional correctness given the performance requirements. A similar test suite will be provided ahead of time, and may be used during to evaluate your project at checkpoints. Some of the tests will not be released prior to the project deadline.

10% Optimality at project due date. This grade is a function of the FPGA resources used by your entire implementation. This score is contingent on a fully functional design (an incomplete project will receive a zero in this category). As discussed in Section 3.5, student designs will be evaluated and binned into a gradient from “optimal” (full credit) to “sub-optimal” (no credit).

10% Timeliness, or functionality at checkpoints. Your project will receive a checkpoint grade at each milestone to help you keep pace. Checkpoint scores will be weighted (details TBA).

10% Design reviews. In the first week of each checkpoint, you must attend a design review, and will receive a grade according to Section 4.1.

10% Final report and style demonstrated throughout the project. Please see Section 4.3 for more detail.

Not included in the above tabulations are point assignments for extra credit, tabulated below:

up to +10% Early completion. See Section 4.6 for more information. In general, point values will be decided by the course staff on a case by case basis, and will depend on the complexity of your proposal, the creativity of your idea, and relevance to the material taught.

6 Project Timeline

This section tabulates when each project deliverable is due (see Table 1). Keep in mind the late policy (Section 4.7) and pace yourself accordingly.
Table 1 CS150 Spring 2010 Master Calendar

<table>
<thead>
<tr>
<th>Checkpoint</th>
<th>Deliverable</th>
<th>Due Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: SMIPS150 Processor</td>
<td>Assigned</td>
<td>2/26</td>
</tr>
<tr>
<td></td>
<td>Design Review</td>
<td>2/28-3/6</td>
</tr>
<tr>
<td></td>
<td>In-lab Checkoff</td>
<td>3/30-4/1</td>
</tr>
<tr>
<td></td>
<td>SVN Submission</td>
<td>3/30</td>
</tr>
<tr>
<td>2: Serial/Ethernet Interfaces</td>
<td>Assigned</td>
<td>3/19</td>
</tr>
<tr>
<td></td>
<td>Design Review</td>
<td>3/30-4/1</td>
</tr>
<tr>
<td></td>
<td>In-lab Checkoff</td>
<td>4/6-4/8</td>
</tr>
<tr>
<td>3: Frame Buffer</td>
<td>Assigned</td>
<td>4/2</td>
</tr>
<tr>
<td></td>
<td>Design Review</td>
<td>4/6-4/8</td>
</tr>
<tr>
<td></td>
<td>In-lab Checkoff</td>
<td>4/20-4/22</td>
</tr>
<tr>
<td>4: Graphics Accelerator</td>
<td>Assigned</td>
<td>4/16</td>
</tr>
<tr>
<td></td>
<td>Design Review</td>
<td>4/20-4/22</td>
</tr>
<tr>
<td></td>
<td>In-lab Checkoff</td>
<td>4/27-4/29</td>
</tr>
<tr>
<td>5: Optimizations</td>
<td>Assigned Due</td>
<td>4/23</td>
</tr>
<tr>
<td></td>
<td>Same as final checkoff (see below)</td>
<td></td>
</tr>
<tr>
<td>Early Checkoff</td>
<td>In-lab Checkoff</td>
<td>4/27-4/29</td>
</tr>
<tr>
<td></td>
<td>SVN Submission</td>
<td>4/27</td>
</tr>
<tr>
<td>Final Checkoff</td>
<td>In-lab Checkoff</td>
<td>5/4-5/6</td>
</tr>
<tr>
<td></td>
<td>SVN Submission</td>
<td>5/4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rev.</th>
<th>Name</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>EECS150 Spring 2010 Staff</td>
<td>3/21/2010</td>
<td>Initial release.</td>
</tr>
</tbody>
</table>