Carlos Cuevas & Cody Winkleblack
6.111 Project Proposal v2.0
7 Nov 2018

Laser Cyclops

6.111 Project Proposal

Table of Contents

Detailed Description

Main Objective

The main objective of this project is to continuously have a laser aim at a target by using a camera's feedback to detect the position of a light-emitting diode (LED) located at the center of the target. A filter shall be used in order to only let the infrared light from the LED through, in which the field-programmable gate array (FPGA) board will translate and determine the target's position; from here, this information will be further converted into two angles, and thereafter two pulses, so that the two servos can adjust themselves and aim directly at the target.

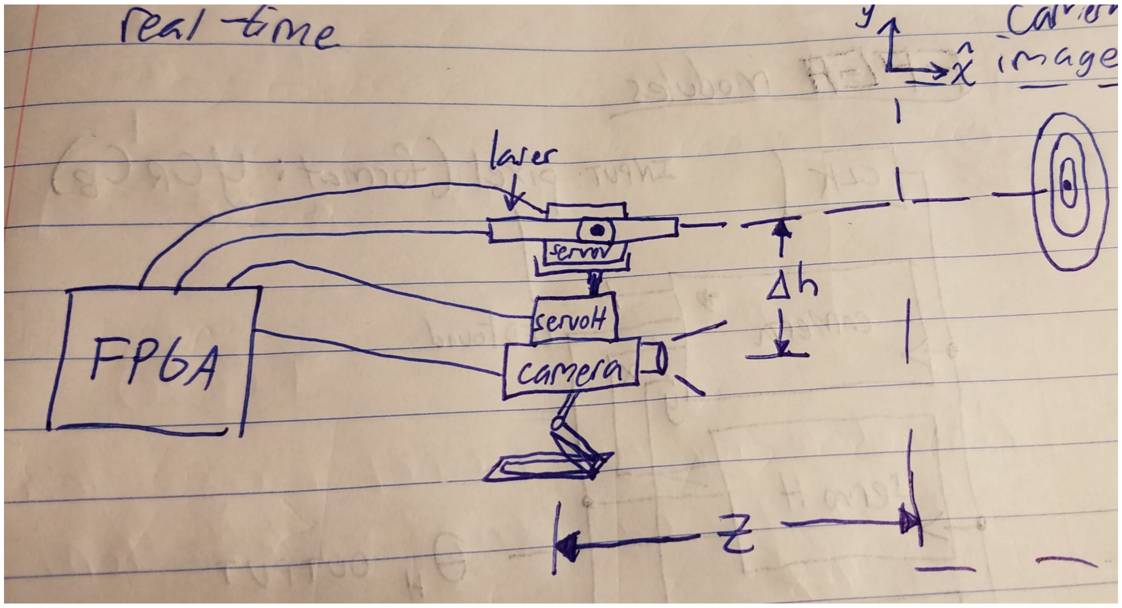

Figure 1: System Overview

Components

The components of the project include:

- Labkit (FPGA)
- 2 servos
- laser
- 3-D printer (for mechanism connecting the servos and the laser)
- NTSC camera
- infrared LED
- target
- infrared filter

The NTSC camera shall be directly connected to the Labkit. This camera sends pixel data in the format *YCrCb*, which will have to be translated to the format *RGB*. This 30-bit wide pixel shall be cut down to 18-bits, in order to best use the ZBT memory; two pixels are stored in every memory address.

Once all pixels in a frame have been saved, the interpreter module shall determine the position of the infrared light and send these coordinates to the modules that will calculate the angle for each servo.

An infrared filter shall be placed in front of the camera. This will only allow light from the LED to go through, which will make it easy for the FPGA to determine it's position, since all other pixel values shall be zero.

The laser will ideally have a simple interface that can be wired directly to the Labkit so that its power functionality can be controlled by the FPGA. This laser shall be powered off when the infrared light cannot be found or, extensively, when the servos are moving; the laser shall be on when it thinks it should be hitting the target.

The servos operate at 5V. A control signal shall be sent every 2ms in the form of a pulse, whose duration determines the angle the servo will be at. A neutral pulse will adjust the servo to 90^o; a control signal less than the neutral pulse shall adjust the servo to an angle between 0^o and 90^o; and a signal greater than the neutral shall adjust the servo angle between 90^o and 180^o. The neutral pulse shall be determined from the specification document of the specific servo in use.

Procedure

This outline shall represent a basic approach to reaching the main objective. Some things can be done simultaneously or slightly out of order. This list might also change during execution.

1. Module Refinement
   1. Break-down all the major modules (listed in the appropriate section of this document) to the most basic level. This will allow the most isolated testing and be easier to find buds in the system.
   2. Determine all inputs and outputs for these sub-modules as well as bus-widths.
   3. Determine the propagation delay per sub-module and/or its complexity.
2. Hardware specifications
   1. All specifications for all pieces of hardware being used should be known or written down. This includes all items listed in the "Components" section above.
3. 3-D printed objects
   1. Preferably using a computer-aided-design program, create an infrastructure to hold the laser, the servos, and possibly the camera altogether. The specifications for all parts that shall be connected must be known by this stage.
   2. Print the infrastructure using a 3-D printer.
   3. Connect all pieces together and test integrity of whole structure (rigidness, balance, etc.).
4. Write Verilog for all sub-modules (this is the largest step)
   1. Knowing all sub-modules that is required, being writing Verilog for each one.
   2. Test each sub-module independently.
      1. Examples include: servo can be moved to an angle inputted on the Labkit's switches, servo can be moved given a coordinate value, camera feed can be seen on monitor using ZBT memory.
   3. Slowly combine modules together and test.
   4. Infrared light should be the last thing to be introduced and tested.
5. Combine entire system and test
   1. Since all sub-modules should have been tested by this point, testing should focus on meeting the main objective.
   2. Testing should also attempt to break the system by: hitting the limits of the camera, removing the infrared light at random instances, etc.

Block Diagram

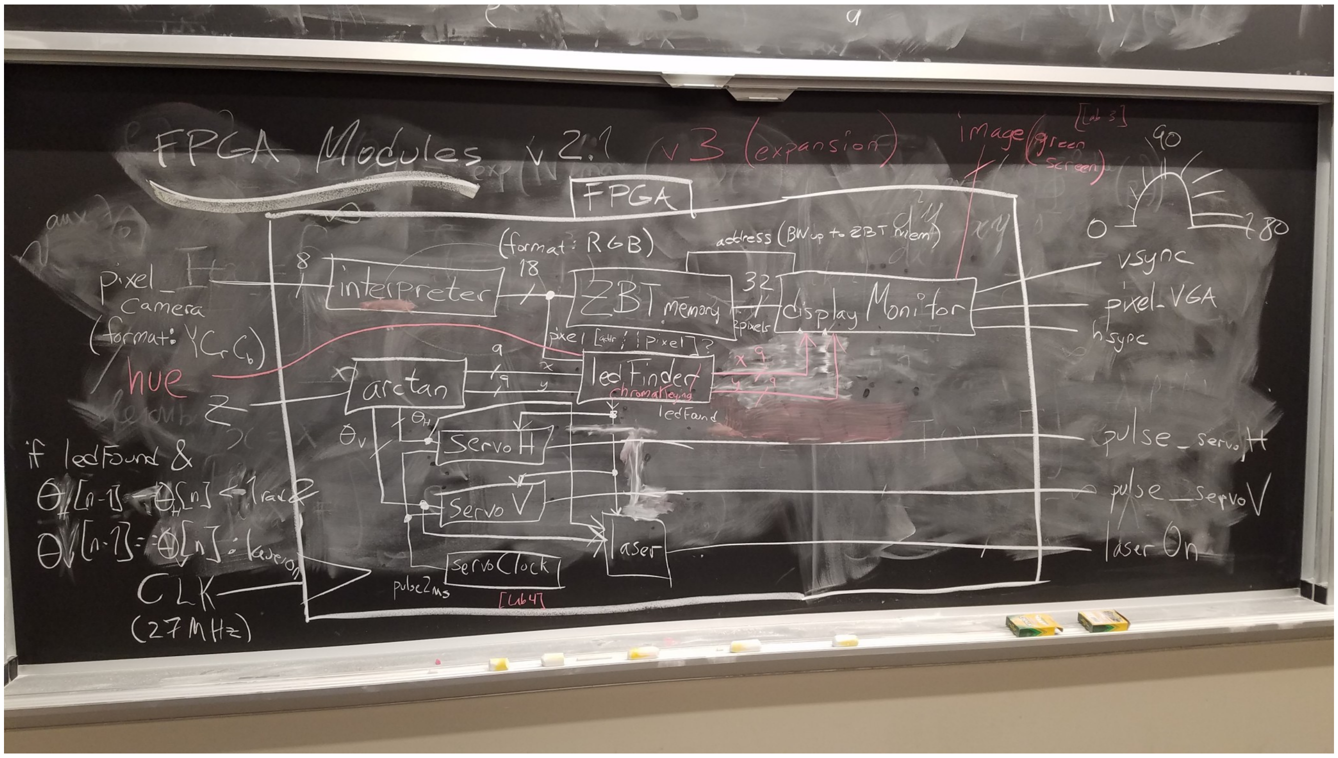

Figure 2: FPGA Block Diagram

Figure 2 shows the block diagram for the FPGA. In it are the major modules *CLK*, *interpreter, zbtMemory*, *displayMonitor*, *ledFinder, arctan servoH*, *servoV*, *servoClock* and *laser*. The following section shall go into more detail, however, do note that the only input shown is from the NTSC camera *pixel_camera* and the depth *z*. There are outputs for the monitor, the two servos, and the laser.

Major Modules

For each major module listed, the following shall be determined: the input(s) and output(s), functionality / objective, complexity, level of performance, and how it shall be tested. Most, if not all, of these modules must be broken into sub-modules. An additional module, called *arctan* will also be inserted in between *camera* and *servoH/V*, which shall be tasked with translating a coordinate to an angle given a depth *z*.

clock (CLK)

- **Input(s)**: None
- **Output(s)**: *CLK* (a pulse at a frequency of 27MHz; this goes to all modules)
- **Objective**: provides the system’s clock. Do note that there’s a separate clock on the NTSC camera (syncing should be verified)!

interpreter

- **Input(s)**: *CLK*, [7:0] *pixel_camera* (coming from the NTSC camera in the format *YCrCb*)
- **Output(s)**: [17:0] *pixel_translated* (format *RGB*)
- **Objectives**: taking the input provided by the NTSC camera, a pixel is given in the format *YCbCr*. This module must then translate this pixel into the format *RGB*. Since there will be 30 bits generated after this translation, this must be cut down to 18 bits using the most significant bits from each color value so that two pixel values can be inserted into each ZBT-memory address.
- **Complexity**: this is a fairly simple module as it is merely translating pixel formats using several multiplications and summations.
- **Testing**: this module should be testing the accuracy of pixel format translation. Several hues should be inserted into the module in *YCbCr* format and given out in *RBG* format. The length of the output must also be verified.

zbtMemory

- **Input(s)**: *CLK*, [17:0] *pixel_translated* (format *RGB*), *address*
- **Output(s)**: [31:0] *pixels* (2 pixels in format *RGB*)
- **Objectives**: should be saving the pixel that is coming in from the *interpreter* module into an address on ZBT memory. Two pixels per address should be saved. With a given address, as

requested by the *displayManager* module, the respective two pixels in that address will be outputted.

- **Complexity**: this is a very simple module, as it is simply used for wiring and keeping a counter.
- **Testing**: this module should be tested for address accuracy, verifying that two pixels are stored per address. The output should also be verified accurate, based on the address inputted.

displayManager

- **Input(s)**: *CLK*, [31:0] *pixels* (2 pixels in format *RGB*)
- **Output(s)**: *address*, [9:0] *hsync,* [9:0] *vsync*, [17:0] *pixel_out*
- **Objectives**: responsible for sending a pixel to the VGA monitor for every (*hsync*, *vsync*) value. It shall extract two pixels for every address request sent to the *zbtMemory* module.
- **Complexity**: although not complex in mathematical operations, this module will be counting up to 250,000 pixels for every frame, at 60 frames per second. It will also need to know that two pixels are stored in every address in ZBT memory.
- **Testing**: similar to Lab 3, observing the VGA monitor will allow us to verify the accuracy of the video feed being sent by this module. Additionally, using the hex display or cross-hairs to lock on the center of what the FPGA thinks is the target will allow us to verify the accuracy of the *ledFinder* module.

ledFinder

- **Input(s)**: *CLK*, [17:0] *pixel_translated* (format *RGB*)
- **Output(s)**: [8:0] *x*, [8:0] *y, ledFound*
- **Objectives**: this module is tasked with determining the center coordinates of the target at every frame, using the pixels that goes through it.
- **Complexity**: a fairly simple module as it consists of several trackers (determining the min and max per coordinate) and division by two (1-bit shifts).
- **Testing**: given a set of pixels, whether via the camera or manually inputted, the accuracy of the center coordinates will be verified. The best method to do this would be to place a marker on the VGA monitor of where the FPGA thinks the center of the target is and visually checking if this is accurate. Samples with a considerable amount of noise should be introduced as well to make sure that noise does not dramatically skew the center coordinates.

arctan

- **Input(s)**: *CLK*, [8:0] *x*, [8:0] *y*, [8:0] *z*
- **Output(s)**: Θ_H, Θ_V (in radians; signed)
- **Objectives**: a pre-written Verilog module found in ISE, this module will take in the two shorter sides of the triangle wishing to compute an angle for (in radians). For this project, the inputs will be a coordinate (*x* or *y*) and depth (*z*), which will be manually given via the switches on the Labkit. The following equations show this:

$$\Theta_H = \arctan\left(\frac{x}{z}\right) \quad \text{and} \quad \Theta_V = \arctan\left(\frac{y}{z}\right)$$

- **Complexity**: a fairly complex module as there is multiplication and a real division calculation in which a bit shift cannot be used (but can be if an estimation is used).
- **Testing**: many different coordinate values should be inputted, as well as those that would be out-of-bounds to make sure that this module handles it correctly; the outputted angles should be verified for accuracy.

servoH | servoV

- **Input(s)**: *CLK*, (Θ_H | Θ_V), *ledFound*, *pulse_2ms*
- **Output(s)**: *(pulse_servoH | pulse_servoV)*
- **Objectives**: given the radian angle for a plane, this module will translate this angle to the appropriate pulse required by the servo. This pulse must be sent every 2ms, which is what the input *pulse_2ms* is for. However, should *ledFound* be low (in which the target is not found), the module should ignore the angle input and move the servos to rest (90^o), using the neutral pulse value.
- **Complexity**: the complexity is very low as this is mostly a look-up table that converts an angle to a pulse duration, which is repeated every 2ms.
- **Testing**: given random angles in the radian domain $[0, \pi]$, the servo should be verified that it is actually pointing in the given angle. At first, each servo should be tested independently and thereafter both simultaneously.

servoClock

- **Input(s)**: *CLK*
- **Output(s)**: *pulse_2ms*

- **Objectives**: a clock that uses the FPGA base clock and a counter in order to send out a pulse every 2ms.
- **Complexity**: a very simple counter module.
- **Testing**: this pulse can be verified by either flashing a LED on the Labkit every pulse or incrementing the hex display at every pulse. If visualizing a 2ms pulse is too quick, this can be extended to pulse the LED every second (for every 500 pulses received).

laser

- **Input(s)**: *CLK*, *ledFound*, Θ_H, Θ_V
- **Output(s)**: *laserOn*
- **Objectives**: this module determines when the laser should be firing. Ideally, this should happen when the following conditional is met:

```
if(ledFound &&
        abs(ΘH[n-1] – ΘH[n])<=E &&
        abs(ΘV[n-1] – ΘV[n])<=E)
    laserOn <= 1;
else laserOn <= 0;
```

  The above conditional simply checks that (1) *ledFound* is high and (2) neither servo angle has changed (between the previous and current time steps) more than a specified error *E*. This error value will most likely be determined through testing but will be about 0.02 radians.
- **Complexity**: a fairly simple module as it's merely computing the above conditional on every clock cycle.
- **Testing**: this can be visually verified by checking the laser stays on when small deviations of the target occurs and turns off when the target is being moved or the LED disappears.

Extensions

Should time allow it, this project can further be extended after the initial main objective is reached with the ideas found in this section. They are listed in chronological order, to which is deemed a logical order to tackle these extensions, as they might build on top of another.

Chroma-keying

Rather than track an infrared LED over a filter (in which the only non-black pixels captured on the camera will be the LED), this extension will remove the filter and use a given hue to lock onto an object whose color falls in that domain. This way, any object with a distinct color can be tracked. A hue will encompass a fairly large range of RGB values so that noise and lighting effects are minimized.

Cross-hair Locking

Rather than having to manually insert the desired hue into the Labkit via the switches, a cross-hair can be laid over the VGA display so that a user can select the desired color from the pixel the cross-hair is over. The cross-hair would be controlled by the Labkit's UP, DOWN, LEFT, and RIGHT buttons.

Image Overlay

This extension would add a pregenerated image, such as the Death Star, and center it over the target so that it seems that the laser is firing at the Death Star. This image can be swapped with a post-laser image, such as the destroyed version of the Death Star after the laser is fired. Since this would only be changing what is seen on the monitor, a marker that is linked with output *laserOn* would be placed over the image to mark where the laser is theoretically firing at.

Cyclops vs Human

This complex extension would allow a user wearing a target on his chest or over his head to challenge the Laser Cyclops by seeing how long he can avoid the laser from hitting him. The color of the target would be pre-inserted into the FPGA. A receiver would be placed on the target that would calculate how long the user was hit; alternatively, the FPGA could count how much time *laserOn* was on.

The user can also be given a certain amount of time to survive the Cyclops and a certain amount of hit points (HP) to retain. A health bar can be displayed on the monitor, showing the user's HP.