ECE532 Final Report

Group 7: Camera Pong

Jazim Akbar

Yogen Krishnapillai

Henry Prickett-Morgan

Hao Yang

Overview

Camera pong is a fully-in-hardware ping-pong video game controlled via camera using a standard red ping pong paddle. The goals were to create a fun, fast-paced, and immersive game experience that would be intuitive to new users and require very limited setup. Camera pong uses an OV7670 PMOD camera as its primary control input and 640x480 VGA at 60fps for output. The system consists of four fully custom IP cores:

- A camera capture and control unit which interfaces with the OV7670
- An image processing unit which cleans up and extracts features from the captured image
- A physics core which simulates the game's physics
- A graphics rendering and VGA output core which renders the full scene and outputs to VGA

A block diagram for the system is shown below:

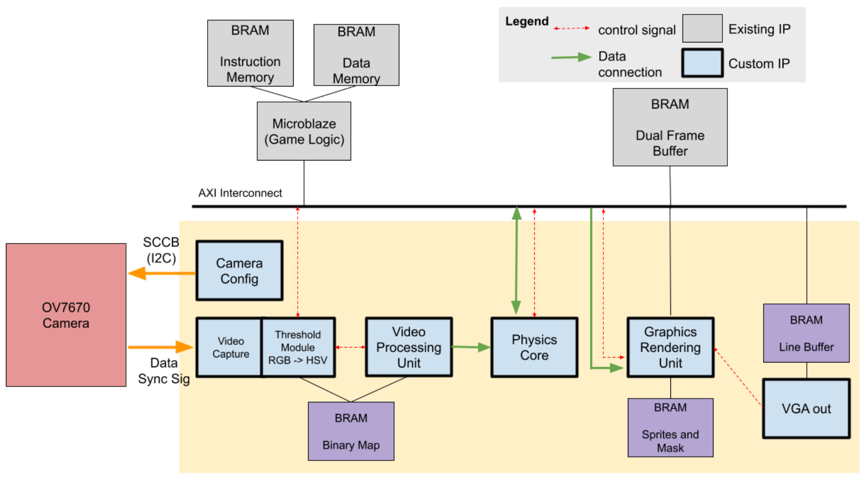

Fig. 1. System block diagram

Outcome

Ultimately, the final system was unfinished. All four major IPs were proved in hardware, but due to a lack of time we were unable to integrate them for the final demonstration. In our demonstration we showed two main capabilities, which we called the “frontend” and the “backend”. The frontend consisted of the camera capture and control unit, the image processing unit, and a custom debugging IP which could output a binary image to VGA. The backend consisted of the physics IP with values specified via microblaze connected to the rendering and output IP, to visualize the results of the physics simulation. There were issues with the interface between the frontend and backend which could not be resolved before our demonstration. The system could be improved by finalizing the integration between the frontend and backend, allowing an entire game to be played via the camera. Were we to start over we would:

1. **Be more careful about warnings and timing errors**:
    a. Standardize all resets to synchronous posedge, except as needed for AXI, and include RDC circuitry when crossing reset domains.
    b. Include CDC primitives on all registers which cross clock domains, rather than as a reaction to bugs within the design.
    c. Ensure that each new block synthesizes and implements without warnings, fixing (or suppressing if really needed) all superfluous warnings so that it is clear when new issues are introduced.
2. **Standardize interfaces early**: Each block expected a different combination of number formats (pixel coordinates, scaled coordinates, integers, fixed point numbers), resets,

clocks, and control signals (pulses versus levels) which added significantly to the complexity of the project. Most of these differences were unnecessary, and could have been avoided by standardizing and designing interfaces at the beginning of the project.

3. **Integrate earlier**: Each block was developed in a siloed environment and tested individually. This allowed fast and frictionless progress early on in the project, but ultimately led to significant integration issues later on. Starting from scratch, we should have integrated the top level system as early as possible and mocked out the interfaces between components.

To take over this project, the most important part would be fixing the remaining interface between the image processing subsystem and the physics IP. To debug remaining issues, it would be best to mock the output format of the image processor (16 bit size, 0 to 639 x-coordinate, 0 to 479 y-coordinate, valid signal lasting roughly ~0.25 ms) and then use that to simulate the behavior of the physics IP, identifying where the breakdown in communication is occurring. Beyond that, adding further features to the design would likely be hindered by the large volume of warnings and timing errors, so a new maintainer would need to invest time to clean up those existing problems and refactor the existing design to make it easier to add further features.

Project Schedule

The following is the original timeline for milestones as planned in our proposal:

1. Jan 31st
    a. Implementation of the image recognition algorithm in Python OpenCV, using only filtering and convolution operations.
    b. Preliminary research of the collision algorithm used for the physics core.
    c. Resource estimates (BRAMs, DDR, LUTs) for the various hardware blocks on the board.
    d. Research VGA rendering.
2. Feb 7th
    a. Implementation of collision algorithms in software.
    b. Specify the top level interface of each hardware module and begin creation of hardware level testbenches for each core.
    c. Begin hardware implementation of video processing core.
3. Feb 14th
    a. Demonstrate datapath connection between video input and video processing core.
    b. Begin rendering simple scenes, such as the table with the paddles and ball in a fixed position.
4. Feb 28th
    a. End-to-end processing of video signals from input to detection of the paddle.
    b. Physics core handling ball movement and collision with the table in hardware testbench.
5. Mar 13th
    a. Physics core handling ball movement and collision with the paddles in testbench.

6. Mar 20th
    a. Full scale integration of physics core, video processing core, and microblaze game logic. Demonstrate a reset-able game mode where a ball is fired from a predefined position and the player can return with a racket.
7. Mar 27th
    a. Development of the AI player in microblaze, demonstrates a volley between the player and the AI.
    b. Full project is completed except for game state controlled in the microblaze.

The progress documented throughout the project within our milestones is as follows:

1. Jan 31st
    a. Preliminary algorithm for detection and location of the ping pong paddle prototyped using OpenCV in python.
    b. Preliminary research of the collision algorithm used for the physics core.
    c. Research VGA rendering and finalize the resolution and framerate for the design.
2. Feb 7th
    a. Complete architectural design of image processing core along with initial FSM design and timing and resource estimates.
    b. Specify initial design for physics of the game, and how it will be handled in hardware.
    c. Research and specify interface with camera for image capture.
    d. Develop ping pong line buffer design and VGA output core.
3. Feb 14th
    a. Simulation verification of the first part of the image processing core.
    b. Test displaying a sample frame buffer to VGA output.
    c. Begin RTL of physics core.
    d. Complete setup and configuration module for OV7670 camera.

4. Feb 28th
    a. Complete implementation of image processing core. Complete simulation verification of the second part (coordinate accumulation) of the core.
    b. Complete RTL and initial simulation testing of physics core.
    c. Complete software reference implementation of graphics rendering. Begin work on defining architecture and capabilities of rendering core.
    d. Complete RTL for video capture and camera configuration with some initial tests.
5. Mar 13th
    a. Debugging, hardware testing and initial integration of video capture with VGA output.
    b. Create a demo project that performs thresholding on input image and displays thresholded image via VGA output.
6. Mar 20th
    a. Test refined HSV thresholding.
    b. Integration of video capture with image processing, with a lot of debugging work done to fix some camera configuration problems.
    c. Update physics core design and rendering core design with new simplified requirements. Specify and implement Axi-lite interface for communication with microblaze.
7. Mar 27th
    a. Complete integration of video capture and processing. Perform some cleanup work packaging different components and creating a final combined project with microblaze.
    b. Verified rendering logic and physics integration in simulation.
8. April 3rd
    a. Working physics and rendering, along with VGA output showing rendered scenes.

b. Working video capture and image processing with clean thresholded object tracking.

Our original schedule was quite ambitious and optimistic. We aimed to have initial implementations of all custom hardware components complete within a month, though this did not end up being the case. We found that for every component, it took much longer to first do the research required for it, then come up with a suitable architecture and translate that into RTL. Both simulation testing and hardware testing was also much slower than expected due to the nature of the components being designed. A lot of the time, what mattered was what happened in between frames which required very long simulations, or otherwise careful hardware test setups.

There were also many unexpected roadblocks that we faced along the way that were not anticipated in the original schedule. One of these roadblocks was the difficulty in setting up and interfacing with the camera. Setting up the camera required finding and providing several setup values that controlled things such as gain, color correction, and exposure correction. It additionally required some magic numbers to provide a good looking output image. Finding settings that worked well was difficult and time consuming, and still did not result in the best image quality. Furthermore, the thresholding parameters needed to be tuned which also took a lot of time. All of these roadblocks pushed back our development time considerably. While we did initially plan several weeks for integration, the problems mentioned above reduced the amount of time we had for integration down considerably, leaving us with little time to fix the inevitable bugs that popped up.

Description of the Blocks

OV7670 Camera and Pre-processing Unit

There are two sub-units in this part, the OV7670 Camera (config & capture) part and the HSV thresholding part. The Fig.2 below shows how these two parts are organized.

As a popular camera model, OV7670 has many online tutorials to work with. We also started our project by looking at how other developers configured and used this camera. Our main reference GitHub repo is https://github.com/westonb/OV7670-Verilog/tree/master. It is a good reference but of course, can not run directly.

We start with reading the datasheet of OV7670 and understanding how SCCB works. The SCCB protocol is very similar to I2C, only the tri-state mode should be noticed. There is another thing that the datasheet doesn't mention. We should add the camera address as 8'h42 at the beginning of each data transfer through the SCCB interface. There are lots of other similar findings that we can only get after careful working. We also use the oscilloscope to track the output pins from SIO_C and SIO_D pins. There is the case that both our simulation and ILA track work but the camera is not configured at all. The problem is that the tri-state signal is not set properly.

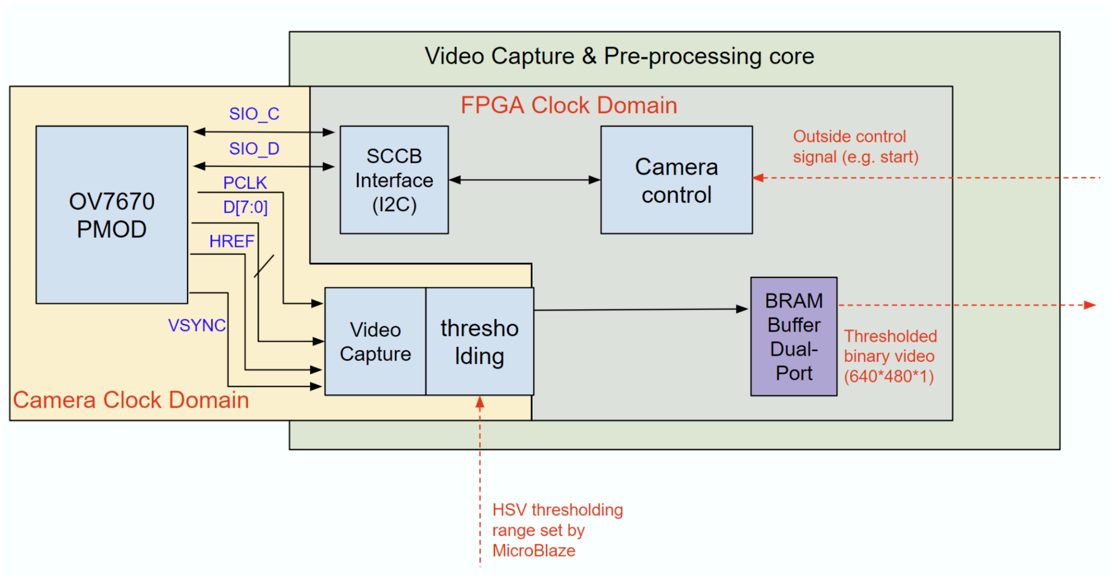

Fig. 2. Camera & Pre-processing units block diagram

Our configuration is chosen based on the options of the datasheet and our needs for this project. For example, with the limit of BRAM resources, we can not save a full RGB565 frame into a frame buffer and display it on the VGA monitor (this is how we prepared the mid-project). We then change the OV7670 output data format into RGB444 to implement it. We also tuned each colour channel's gain to capture our paddle at the early stage better when we were using the RGB thresholding.

The HSV thresholding part is a fully custom IP. The formula for how to calculate HSV from RGB is easy to get. The difficulty is how to complete the division operation. Our system becomes robust after introducing the HSV thresholding. HSV color space is popular in the computer vision area because of its accurate color distinguishing ability. In the end, we also set the port from MicroBlaze to the HSV thresholding part. We can tune our thresholding range from SDK and manually make it adjust to different light conditions.

Image Processing Unit

This is a custom image processing IP which operates on a binary image stored in a 640 by 480 by 1 BRAM and calculates the position of the target within the image.

Port	Direction	Size	Description
clk	Input	1	100 MHz input clock.
rst		1	Posedge synchronous reset.
image_valid		1	Valid signal for the image contained within the BRAM. Must be held high while the image is still valid, at least 1.5 ms.
bram_douta		640	Read from the BRAM to the image processor.
bram_dina	Output	640	Writeback from the image processor to the BRAM
wea		1	BRAM Write Enable
addra		9	BRAM Address
ena		1	BRAM Enable

p_x		10	Calculated x-coordinate of the target [0,640)
p_y		9	Calculated y-coordinate of the target [0,480)
p_size		16	Calculated size of the target.
coord_valid		1	Valid signal for p_x, p_y, and p_size.

Table 1. I/Os design of imaging processing unit

The IP works in two stages. The first stage is handled by the erosion_dilation unit. During this stage, the IP will buffer three rows of the input image starting from the top and use that to calculate a new row via erosion or dilation. This involves performing a 3x3 convolutional-and (erosion) or 3x3 convolutional-or (dilation) on the input pixel values. This new calculated row is then written back to the BRAM and a new row is read in. Reads and writes are interleaved (writes lag reads by three rows) so that the operation can happen in place without overwriting data that will be used in the future. After 6 erosion passes and 6 dilation passes, the erosion_dilation unit is finished and the image is ready to be analyzed.

The second stage is the pixel accumulator unit. This unit contains three dot-product-and-accumulate units (named vector_mac in the hierarchy). The input image is scanned over in 32 bit chunks and three coefficient vectors are maintained. The x-coefficient vector contains the x coordinates of each pixel in the 32 bit chunk (so for the first 32 bits of the image the x-coordinate vector would be [0, 1, 2, 3, 4, 5, …, 31]). The y-coordinate vector contains 32 copies of y-coordinate of the chunk, which is just the row index. The z-coordinate vector is a copy of the 32 bit input, as the "z" coordinate is just the number of pixels which are "on" in the input image. For each chunk, the pixel vector and coordinate vectors are fed into the vector_mac, which is a fully pipelined dot product accumulator. After the entire image has been scanned, these accumulators will contain the sum of the x coordinates of all pixels in the image, the sum of the y coordinates of all pixels in the image, and the number of on pixels in the image.

As a final processing step, the x and y coordinates are fed through a Xilinx divider IP where they are divided by the z-coordinate to give the average x and y coordinates of "on" pixels in the image, which is considered as the center of the target. The coord valid signal is delayed according to the fixed delay of the divider IPs.

The testbench for this unit included both Python and SystemVerilog components. I wrote a script to dump a binary image from the Python OpenCV library to a CSV file, which could then be loaded by a SystemVerilog testbench. This testbench would load the image into the BRAM, run the processing step, and then dump the processed image. The output signals of the modules (p_x, p_y, p_size) were also dumped by the testbench. The processed image and output coordinates were then compared against the software implementation for correctness.

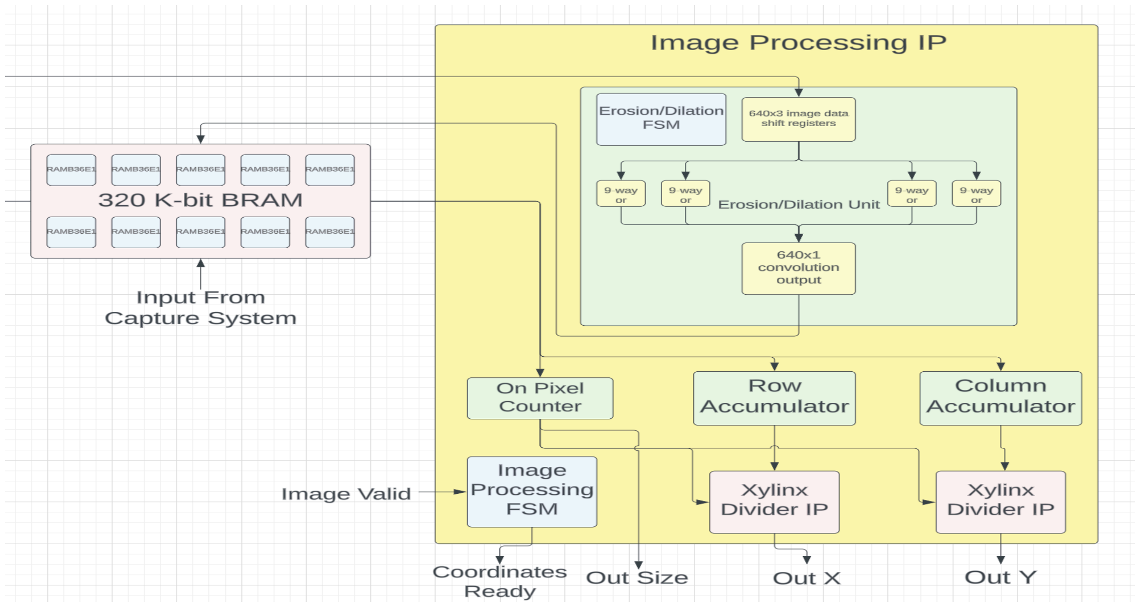

Fig. 3. Block diagram for the image processing subsystem.

Physics Unit

The design of physics IP was developed by identifying the Control and Data path logic for the physical aspects of the collision between Paddle, Ball, Net and ping-pong board. The most challenging part of this design is to grasp the requirements correctly and plan the exact complexity and implementation of the physics and collisions. As the first step, all the technical logical information was gathered, studied and documented.

First, identifying the Clocking Scheme, Reset Methodology and Interface ports are the key component of a successful design of Physics Core. The Paddle's X, Y, Z Coordinates and their bit sizes and the corresponding valid signal were determined by consultation with the team members. Since both Physics Core and Image Processing modules are functioning at the same clock source, it was determined that the clock domain crossing would be fine as long as the clock is coming from the same source and have the same phase relationship. In general practice, the FPGA design tends to choose Synchronous Reset Methodology. But for the physics core design, the Asynchronous Reset methodology was chosen due to the asynchronous nature of source inputs and reset was given higher priority during the operation.

The control logic for the physics core was developed through Finite State Machine (FSM) logic. As per the FSM RTL coding style guideline, the FSM coding can be either Three-Always or Two-Always or One-Always depending on the complexity and usage of the design. For simplicity and efficiency, One-Always FSM was chosen to handle the control logic of our Physics Core design. The FSM was developed to address the main functionality which included calculation of Paddle Velocity, Ball Velocity, detection of Ball Collision, detection of Table Collision and detection of Net Collision and the calculation of Ball's Position.

To keep the design simple, axis-aligned bounding boxes were used for all game objects. This allowed collisions to be detected as overlaps between two boxes, and since the boxes are aligned, this only required comparison of the position and dimensions. There were three main parts of the FSM, performing updates of the ball position and velocity, checking for colossians, and properly handling the collisions. The ball position was first speculatively updated and that new position was used to check for collisions with the ball and all other game objects. Upon a collision, the FSM transitioned into the associated collision handling state, which could be tuned for each type of collision. The collision handler determined the new velocity of the ball. At the end, this new velocity was used to perform the update on the ball.

The Datapath logic that requires addition, subtraction and multiplication were done through combinational logic. The important initial values for Paddle, Table, Net, Ball were done through localparam assignments. To facilitate game control, the current physics game state was accessible by the Microblaze via Axi-lite interface. The Microblaze was also able to write to control registers to run and reset the physics core.

Rendering and Output Unit

The rendering unit is the component that takes the coordinates of the game objects from the physics space, translates these coordinates to the corresponding positions on the display, and renders the game sprites to their appropriate position and size.

The sprite for each game object is stored in BRAM as a 4 channel image. The image contains the RGB values, as well as an additional mask bit that indicates whether or not that pixel should be rendered. The X, Y coordinates of the object determine the corresponding position of the sprite on the screen whereas the Z coordinate determines the scale of the object (the size as seen on the screen).

Two framebuffers are stored in BRAM, which act as ping pong buffers between the rendering and the VGA out units. While one frame is rendered, the other is displayed. Before rendering any of the sprites, the screen is cleared and all of the sprites are rendered in order from furthest away from the screen to closest to screen, with the order being determined by the Z value of the object coordinate. To achieve timing requirements, a framerate of 60FPS, AXI-bursts are used to write each line of the sprite. The read address for each pixel is generated as a function of the Z position, and a scaling factor that is unique to each sprite. This operation requires a floating point division, though to simplify the operation an estimate is used where the division operation is split into a multiplication and a division by a power of two, which can be performed as a bit shift.

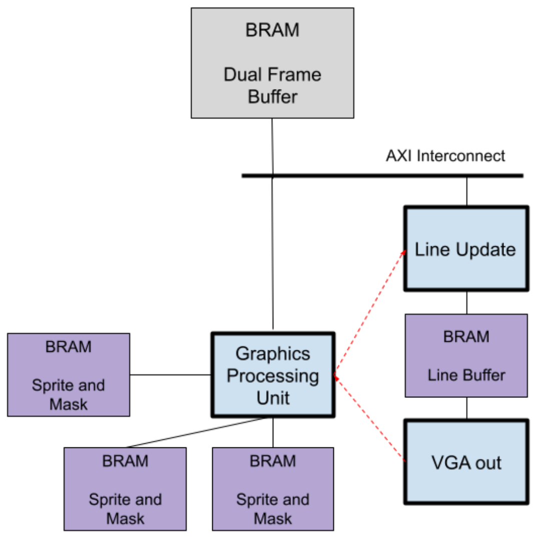

Figure. 4. Diagram of rendering and output unit

Microblaze

The microblaze served as the communication layer between the physics and rendering IPs, as well as a configuration service for the image thresholding and physics IPs. After a reset, the microblaze will program configuration registers in both the thresholding and physics IPs to set up tuned default values for the HSV threshold, ball initial position and velocity, and control registers for the physics simulation.

After the design is configured, the microblaze will repeatedly poll the physics IP for new physics results. For each renderable object (both paddles, ball, and the table), the microblaze will take the coordinates given by the physics IP and compute a perspective transform to place them in the coordinate system of the virtual "camera" used in the game. These camera coordinates, as well as scaling factors allowing us to dynamically resize sprites, are then written via AXI to the rendering IP, to be rendered during the next frame.

Description of Design Tree

Our project design tree is well organized and you can find the details below:

- **ROOT:** the top-level directory contains all other sub-folders
- **bd/:** block design: contains the memory initialization file and block design top wrapper
- **board/:** contains the board file from Digilent. Our project is based on the Nexys 4 DDR Board. Please make sure you choose the correct board file before running the tcl to create a project.

- **doc/:** contains our group report and final demo slides
- **ip/:** contains the IP files for this project. Including the existing IP for DDR and our custom-packaged IP.
- **prj/:** this is the folder where you can start a new project with tcl.
- **sdk/:** contains the C file used for MicroBlaze application.
- **sim/:** contains the testbench we used for debug
- **src/:** contains all the RTL design files
- **xdc/:** contains the constraints files

Tips

Based on the team's experience, we all went through different types of experience from the start to the end of the project. Here are some lessons that might be helpful for the future students of this course to make your team-work more memorable and successful.

First, the selection of project topics of what you plan to design is extremely important. It would be advisable to spend enough time to explore the various design proposals and then choose the most ambitious project that would involve each team member's complete participation as per their expertise.

Understanding the design requirement and technicality behind the design are important key factors for the successful completion of the project. If you spent much less time on architecting the proposed design, then, you would end up spending much more time on debugging the design at the last minute. To overcome the last-minute time-consuming debugging task just days before the final deadline, the advice would be to spend a reasonable amount of time on architecting the design, documenting and having meaningful discussions among team members to clarify the design requirements and technicality of the design as early as possible.

This ECE532 course not only prepares students in technical skills needed for FPGA design, but also prepares students to develop the project management skill as a team to get ready for the real-world experience. Therefore, during the early stages of the project, the task distribution of the entire project should be done very clearly with the full consent of the team members so that each team member knows exactly what design modules that they are expected to complete according to their expertise in Hardware, Software, System Level design and System level verification. This would help to avoid situations where one team member steps into another team member's designated task or avoids taking over the ownership from the other team member.

The sense of ownership would naturally give the sense of achievement for each team member. As a team, make sure that each team member owns one or two design modules as initially assigned at the beginning of the project. During the project implementation phase, one team member may offer suggestions to other team members as part of the team-work which would help to build a friendlier and healthier team. But, the creativity and the sense of ownership for each team member should be preserved and should not be taken away at any point during the project. This would help each team member to feel their hard-earned contribution is counted towards the final product. Every team member should be given equal airtime and visibility during the presentation to convey their individual contribution to the class Professor and the rest of the students. At the end of the day, the sense of ownership and sense of achievement is what each team member would take-away from this course. These tips would help future students of this course to make the right choice and effective decisions to make successful projects.