|  |
| --- |
| zipdt |
| Final Design Report |
| EOS 2012 |
|  |
| **Michael Davis, Michael Driscoll, Yifan Ge** |
| **12/13/2012** |

|  |
| --- |
| [Type the abstract of the document here. The abstract is typically a short summary of the contents of the document. Type the abstract of the document here. The abstract is typically a short summary of the contents of the document.] |

Table of Contents

[Project Overview 2](#_Toc343173415)

[Proposed Solution 2](#_Toc343173416)

[Design Overview 2](#_Toc343173417)

[Tetris Game Behavior 3](#_Toc343173418)

[Design Specifics 3](#_Toc343173419)

[Shared Constraints 7](#_Toc343173420)

[Software Phase Constraints 7](#_Toc343173421)

[Hardware Accelerated Phase Constraints 7](#_Toc343173422)

[Budget 7](#_Toc343173423)

[Camera Selection 7](#_Toc343173424)

[Engineering Process 9](#_Toc343173425)

[Filtering Methods 9](#_Toc343173426)

[Skin color filtering 10](#_Toc343173427)

[Background subtraction 10](#_Toc343173428)

[Object Recognition 10](#_Toc343173429)

[Haar Classification 10](#_Toc343173430)

[Hough Transforms 11](#_Toc343173431)

[Contour Analysis 11](#_Toc343173432)

[Hand Gesture Recognition Methods 11](#_Toc343173433)

[Accelerators 12](#_Toc343173434)

[Memory Sharing 12](#_Toc343173435)

[Tentative Schedule 15](#_Toc343173436)

# Project Overview

The goal of this project is to design an application using the new Zynq development board from Avnet, the ZedBoard, that stimulates interest in using the board for project development. The application will demonstrate two of the original target applications of the board: video processing and hardware acceleration. To demonstrate capabilities of the Zynq platform, this project will perform image processing operations on HD images in real time and accomplish game control with the processed information. In order to reinforce the benefit of hardware acceleration and show the practicality of integrating accelerators using Zynq, the project will be designed in two phases. In the first phase, the application will be implemented solely in software, with the OpenCV algorithms running entirely on the ARM processors. The second phase will take a portion of the algorithms from phase one and utilize the programmable logic (PL) to implement a hardware accelerator. The target audience for the application is potential users in industry who are interested in using the Zynq 7000 family. The portion of the algorithms that is realized in the PL should perform an image processing function that is relative to the target audience.

This document outlines the final design decisions made up to this point, as well as the engineering processes that have brought us here. The project description is covered in a hierarchical manner beginning with application interaction and behavior, design specifications, gesture and motion recognition methods, and accelerator design considerations. Finally, a projected budget and schedule for next semester containing assigned tasks and deliverables are presented.

# Proposed Solution

## Design Overview

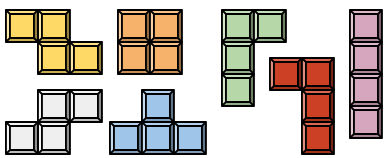
The application is designed as a hand motion and hand gesture controlled Tetris game. We have defined hand gestures as the number of fingers the user has extended on the hand being monitored by the system. Hand motion is defined as the movement of the hand in the columns of pixels between subsequent captured frames.

Prior to the start of gameplay, the system will undergo a short calibration period. During this time the user should remain still and stand in the location that they will play the game from. The user’s hand that will be used to control the game should not be held up during this time. This calibration stage will gather information about the environment so that the system may better extract the user’s hand motion and gestures.

To begin the game, the user will raise their hand that will be used to control the game with their palm parallel to the camera’s image plane. At this point the tetris game will be initiated. The user will be able to then control the horizontal motion of game pieces by moving their hand from side to side in front of the camera. The orientation of each piece will be controlled by changing the number of fingers the user has extended.

### Tetris Game Behavior

The Tetris game consists of seven different tetrominoes, which are geometric shapes composed of four square blocks shown in figure 1. The playing space, or board, is a 10 column by 22 row space (only 20 of which are visible).



**Figure 1.** These are the seven shapes of the tetrominoes used in the tetris game. The colors may change upon implementation.

The user begins a game of Tetris with zero points, an empty playing area, and on the first level of difficulty. At this point, a tetromino is randomly generated for the user at the top of the screen. This piece begins to descend downward in the playing space at a relatively slow speed (due to the lowest difficulty level). At this point, the user is able to rotate the piece by 90 degree clockwise increments, move the piece from side to side within the playing area, or force the piece to drop as far as it can in the playing space. Tetrominoes may not be rotated or moved if the manipulation results in a collision between the user-controlled piece and a previously placed tetromino or the boundaries of the playing area. When a piece reaches the bottom of the playing area or attempts to move downward onto another piece it stops its descent and the user is no longer able to control the piece. When the user loses control of a tetromino, another is randomly generated at the top of the screen and the process is repeated. The game ends when tetrominoes are stacked to the top of the playing area, preventing another tetromino from being generated.

The user is able to score points in the game by filling an entire row in the playing area. When an entire row is filled with blocks of tetrominoes, it is deleted causing all blocks above the cleared row to move downward one space to accommodate for the removed row. Due to the shapes of the tetrominoes, it is possible to eliminate more than one row at a time. These row completions are more difficult, and as such, reward the user with more points. The scoring is as follows: 1 row 40 points, 2 rows 100 points, 3 rows 300 points, 4 rows 1200 points. The level of difficulty is increased with every 10 rows cleared. As the difficulty is increased, the rate of descent of the user-controlled tetrominoes is increased.

## Design Specifics

As the purpose of this application is to demonstrate the capability of ZedBoard and its benefit of having both processing system and programmable logic on the single chip, we decided to impose some limitations on the player to make the Tetris game easier to implement with the available OpenCV library. Some of the limitations address the specification of certain OpenCV functions, while some of them are stated because of the physical layout of the game. The user limitations are listed in Table 1.

Table Limitation of the system

|  |  |  |
| --- | --- | --- |
| Limitation | **Acceptable Value** | **Justification** |
| User hand orientation | Palm surface must be parallel to camera’s image plane | The user’s fingers must be easily distinguished by the system. |
| Allowable user hand movement | Hand must remain in calibrated region | The user must keep their hand within an area that the system will recognize. |
| Users in environment | One user | System may only handle input from one user at a time. |

In this section, we use Table 2 to address the requirements of the game. Some of these requirements have been provided by Avnet. The rest of them are from our interpretation of the game behave based on the research results available online. The detailed justification for each individual element is included in the table.

Table Performance Base Specifiation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Requirement | **Acceptable Value** | **Justification** | **Test Procedure** | **Priority** |
| System gesture recognition time | <1 second | Response time should be short enough so that gesture responses from the system can be recognized | The system will be supplied with a pre-recorded video of a user changing hand gestures. The response time will be the time it takes for the system to recognize the user’s final gesture, after the user is no longer manipulating their hand. | High |
| Minimum number of distinct hand gestures | 3 | There should be enough hand gestures so that the user may have full control over an individual tetris piece. | Footage of a user demonstrating a series of supported gestures will be supplied as input to the system. Correct differentiation of supported gestures will prove the functionality. | High |
| Hand motion detection | at least 2 directions | The system must be able to identify hand motion in 2 directions (i.e. left and right). | Pre-recorded video of a user moving their hand in two directions will be supplied to the system. Success will depend on the system correctly identifying the two separate directions. | High |
| Processed image resolution | 720p or higher | The processed image resolution must be high enough to demonstrate the capabilities of the zedboard. |  | High |
| User input recognition range | 3-5 ft | User should be close enough to the camera to allow distinct finger recognition, while allowing a comfortable distance from the monitor | Video of user displaying supported gestures will be be recorded while the user is at the minimum and maximum distances from camera. This footage will then be supplied to all other test procedures listed in this document requiring recorded video. Success will be determined based on the success of each test procedure. | Medium |
| Percent correct gesture recognition | >70% | The detection algorithm must be able to discriminate user input from the environment. Master’s-based projects have achieved true positive rates between 80% and 100% | Pre-recorded footage of specific correct and incorrect hand gestures will be passed through the gesture recognition algorithms. The resulting determinations of each hand gesture will then be compared to the actual gesture in order to approximate success rate. | Medium |
| Percent correct hand motion recognition | >70% | The detection algorithm must reliably identify the user input. | Pre-recorded footage of specific hand motions will be passed through the recognition algorithms. The resulting determinations of each hand motion will then be compared to the actual motion. | Medium |
| Maximum calibration time | 45 seconds | Some time must be set aside for calibration methods, such as background subtraction. |  | Medium |
| Maximum gesture time length requirement | 1 second | Some time is required in order to perform image processing methods used to determine a specific gesture. We also require several frames to confirm a specific gesture. | Pre-recorded footage of specific hand gestures will be passed through the recognition algorithms. Time will be recorded from when the initial gesture occurs to the when the system outputs the corresponding gesture determination | Medium |
| Output display resolution | VGA or higher | Output resolution must be sufficient for displaying the tetris game. |  | Low |

Besides the performance based specifications, there are some physical constraints listed below that are limited by the ZedBoard hardware specification and the client request.

### Shared Constraints

1. Application implemented using the ZedBoard
2. Utilize the OpenCV libraries
3. Operates using embedded Linux Kernel
4. Software written in C/C++
5. Retrieve images from an external camera
6. System must provide a response to the image data
7. Resource limitations of the ZedBoard must not be exceeded
8. The USB camera used for the application must not exceed $100

### Software Phase Constraints

1. All image processing achieved in software

### Hardware Accelerated Phase Constraints

1. Some image processing achieved in FPGA fabric

## Budget

The final application consists of three important parts: ZedBoard, monitor, and camera. Among these three components, we obtained three sets of ZedBoard kits from Avnet and planned to use the monitor in the laboratory space. Thus, the only part left for purchase is the camera. However, in order to find a camera that functions with ZedBoard and is suitable for our project, we needed to look for camera that satisfies following requirements.

1. The camera needs to be compatible with Linux operating system
2. OpenCV has to be able to use the camera with its function
3. The resolution of the camera has to be at least 720p
4. The price needs to be within $100

### Camera Selection

After searching online, we found that OpenCV has a list of compatible cameras that can be used in the Linux operating system, part of the list is shown in Table 3.

Table OpenCV compatible cameras

|  |  |  |  |
| --- | --- | --- | --- |
| Camera | **Type** | **Driver** | **Tested On** |
| Basler A301/302/311/312 f/c | ieee1394 | video1394 | i686 Linux 2.6 |
| Basler A601/602 f/c | ieee1394 | video1394 | i686 Linux 2.6 |
| Focus Robotics nDepth Stereo Camera | PCI Video Grabber | fr3 (v4l2) | i686 Linux 2.6 |
| Logitech QuickCam for Notebooks Pro | USB | PWC | i686 Linux 2.6 |
| Logitech QuickCam for Notebooks Pro | USB | PWC | x86\_64 Linux 2.6 |
| Logitech QuickCam Pro 4000 | USB | PWC | i686 Linux 2.6.8 |
| Logitech QuickCam for Notebooks Pro (new version) | USB | UVC | i686 Linux 2.6 |
| Logitech QuickCam Fusion | USB | UVC | i686 Linux 2.6 |
| Logitech QuickCam Pro 5000 | USB | UVC | i686 Linux 2.6 |

Since the ZedBoard is equipped with USB OTG ports, it will inexpensive and easy to use a USB camera. From the list in Table 3, the USB drivers that are supported by Linux are PWC, which is Philips USB Webcam Driver for Linux, and UVC, which is USB Video device Class driver for Linux. However, PWC only works with VGA resolution cameras. Since our project needs 720p or higher resolution, we decided to select a camera that uses the UVC driver. Besides the drivers, we also noticed that many logitech cameras are compatible with OpenCV under linux OS. With this understanding, we started the camera search with logitech. We searched for cameras with at least 720p resolution and less cost less than $100. Table 4 lists the cameras we found satisfy these criterias.

Table Potential cameras list

|  |  |  |  |
| --- | --- | --- | --- |
| Camera | Logitech C920 | Logitech C615 | Logitech C310 |
| UVC support | Yes | Yes | Yes |
| Field of view | 78 degrees | 74 degrees | 60 degrees |
| Video Resolution | 360p, 480p, 720p, 1080p | 360p, 480p, 720p, 1080p | 360p, 480p, 720p |
| Maximum Frame Rate | 30FPS@1080p | 30FPS@640 by 480 | 30FPS@640 by 480 |
| Price | $74.99 + Shipping | $54.86 + Shipping | $35.00 + Shipping |

From the list above, we decided to purchase Logitech HD Pro Webcam C920. This camera utilizes a UVC driver allowing it to interface with the Linux UVC driver. The C920’s predecessor, the Logitech C910, has been successfully tested with in OpenCV on a Linux OS. Therefore, we this webcam should work with ZedBoard using the Linux UVC driver. A useful feature of the Logitech C920 is the ability to select different resolutions, varying from VGA to 1080p. Since we haven’t decided the final resolution for the application, this flexibility will allow us to do different testing with ZedBoard at different resolutions. Finally, the price for Logitech C920 is $80.03, which is well within the budget of $100. With these justifications, we think that Logitech HD Pro Webcam C920 will be a good selection for our project.

(source: <http://opencv.willowgarage.com/wiki/Welcome/OS>)

After we selected the camera, we created the budget table as shown in Table 5.

Table Budget table for the project

|  |  |  |
| --- | --- | --- |
| Item | Quantity | Price |
| Logitech C920 Camera | 1 | $80.03 |

# Engineering Process

Because of unspecified application, we were unable to determine the detailed specification of the application at the beginning of the semester. Instead, we started with preliminary research to obtain a better background knowledge on the capability of the ZedBoard and implementation methods of image processing.

The project is divided into three independent sections between our team members: OpenCV, ZedBoard, and programmable logic implementation. Following initial background research, each member of the team focused on learning the fundamental concepts and developing intermediate demonstrations in their focus areas. With a better understanding of the necessary background knowledge, we have combined what we understand from our learning experience to design our final application. In this section, we will go in depth on the details and present our design considerations of our project. These considerations are given based on research completed during this semester.

## Filtering Methods

Most image processing applications involve some form of image filtering in order to isolate specific features of an image. As a result, an investigation of several likely implementations of filtering for our tetris development is warranted to ensure that each method is feasible. Two likely methods of image filtering that would assist in isolating the hand for gesture and motion recognition are skin color filtering and background subtraction.

### Skin color filtering

As as initial method of filtering, skin color filtering can be used to eliminate a significant number of objects from a frame. This is fairly simple to accomplish in software, as the use of a single OpenCV method, cvCvtColor(), can be used in order to effectively isolate pixels in a frame that correspond to a predetermined pixel range. It is important to note that conversion to another color space is commonly performed in order to provide a more accurate color range for skin tone. With OpenCV, we were able to develop a filter that isolated skin using HSV values. Other examples have implemented a similar filter using the YCbCr color space. The YCbCr color space is generally more commonly used than RGB, as RGB is not considered efficient for transmission purposes. More complex methods for skin filtering include the use of a probabilistic model, which involves an algorithmic determination of color range values based on a number of sample images.

### Background subtraction

With the desire to use hand gestures, one aspect that needed to be addressed was the influence of the user’s face on gesture recognition. One possible way to eliminate this potential problem is through background subtraction. OpenCV contains direct functionality to perform background subtraction on a frame. In practical applications, background subtraction should be based on a large number of images in order to eliminate noise from the resulting filtered image. We were able to develop a background subtraction algorithm based on 300 frames, which requires 10 seconds of calibration based on a 30 fps camera. Because we desire less than 45 seconds for calibration time in our final design, we did not desire to implement an algorithm based on more frames, as the frame rate of the program may decrease when utilizing the ZedBoard. Further filtering may be performed on a image after background subtraction in order to eliminate any stray noise or small movement that could potentially interfere with image processing techniques. OpenCV contains functionality to effectively eliminate small regions identified to be in the foreground, as well as join large regions of pixels in the foreground to form blobs.

## Object Recognition

As our desired tetris game requires some form of object recognition in order to determine several unique hand gestures, we analyzed our prior research in order to determine which algorithms could be implemented with OpenCV to perform the desired task. Three object recognition methods that were researched throughout the course of the semester included Haar classification, Hough transforms, and Contour analysis.

### Haar Classification

Haar classification involves analysis of an image based on the use of a Haar classifier .xml file. This .xml file is created through analysis of description files that contain both positive and negative samples of the desired object. OpenCV contains a Haar training utility that may be used to created an .xml file from the aforementioned description files. In a sample implementation, we were able to develop a program that effectively identified faces in a frame. This was accomplished by using one of the facial recognition Haar classifier files that was packaged with the OpenCV library. A possible method for developing our own .xml file would involved storing all available frames from a camera capture and packaging those images into corresponding description files. In terms of hardware implementation, we researched several cases in which Haar classification was implemented with an FPGA. On standard resolution (480p) images, cases involving facial recognition usually took approximately 0.25 seconds. However, this was accomplished at a fairly low frame rate (<10 fps).

(Sources: <http://cseweb.ucsd.edu/~kastner/papers/fpga09-face_detection.pdf><http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04629966>)

### Hough Transforms

For hough transforms, we found that this method could be used to identify the circularity of fingertips and webbing in an individual frame. We could then associate the number of detected circles in an image, as well as their relative position, to a specific gesture. Furthermore, there are many existing examples that provide an outline on how to perform a Hough transform with an FPGA. However, upon researching the algorithm to perform this task in hardware, we found it to be computationally expensive. This is rather costly considering response time is a high priority task for the project. One example of a hough transform, as implemented on an FPGA, resulted in a throughput of 25 fps. This did not appear to be promising, largely due to the fact that the throughput would likely decrease when implementing the algorithm in software.

(Sources: <http://144.206.159.178/ft/CONF/16408621/16408634.pdf>

<http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=886737>)

### Contour Analysis

We also analyzed the potential of determining the contour of the objects in a frame and analyzing said contour to associate the object with a particular gesture. We were able to implement this object recognition method in OpenCV by utilizing particular functions that streamline the process to determine all contours in an image. Prior to contour determination, extensive filtering was required in order to isolate the hand in the frame. Concerning hardware implementation, we again found several examples where contour tracing was performed with an FPGA. From these examples, we found that the feasibility of implementing contour tracing for our project in hardware is fairly high. This is because the throughput of the contour tracing algorithm in hardware, with parallelism, can approach 200 fps. As a result, a contour accelerator would likely improve the performance of our overall system, which is a desirable feature of our project in order to highlights the capabilities of the ZedBoard.

(Sources: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4379791>)

### Hand Gesture Recognition Methods

As our tetris implementation will implement hand gesture recognition, we found it useful to research exact cases where a similar operation was performed in hardware. Two academic papers, which provided a promising example for our final design, involved the use of an FPGA to identify the number of finger an individual was holding up. This was accomplished by isolating the hand initially using a skin-color model. After filtering, the center of mass of the hand was determined using an average of all pixels within the identified hand area. The number of fingers could then be determined by enumerating the number of color transitions at a discrete radius around the center of mass of the hand. This may be applicable to our development of tetris because we could directly relate the number of fingers being held up to the rotational position of a tetris block. If we were to use this method, we would likely implement background subtraction and small area filtering in order to eliminate the user’s face from the camera image, as the skin filter alone would not accomplish this task.

(Sources: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6188677>

<http://www.cs.bu.edu/gradprogram/thesis/SteveCrampton.MA.thesis.2004.pdf>)

## Accelerators

The hardware accelerated version of the project will feature custom hardware accelerators in the programmable logic of the Zynq. The goal of these accelerators will be to increase the performance of an image processing operation. Implementing the hardware accelerators will likely take a significant amount of time and effort. As a result, the portions of the algorithm that are the most computationally intensive or would most benefit from parallelization will be considered the best candidates for acceleration in order to provide the most benefit for our efforts.

The accelerators will be visible in the application space of the PS. To communicate with the accelerators, Linux device drivers will be written to access the peripheral’s control registers from the application space. Use of the open, mmap, ioctl, read, and write system function calls will be good command candidates for interfacing with the hardware accelerator peripheral. In order to control access to the shared frame buffers while the accelerator is operating on the data, semaphores will be used to provide mutual exclusion. Interrupts will be connected to the PS interrupt controller from the PL to signal accelerator operation completion and to post the frame buffer semaphore.

### Memory Sharing

Since the processing system will be reading images from the camera and storing them in memory, the accelerator peripherals in the programmable logic must share a memory range with the processing system. Multiple methods for sharing memory have been investigated over the course of the past semester.

Information about DMAC from:

<http://infocenter.arm.com/help/topic/com.arm.doc.ddi0424a/DDI0424A_dmac_pl330_r0p0_trm.pdf>

The Direct Memory Access Channel (DMAC) peripheral request interface is a feature of the processing system’s DMA controller which allows CPU-visible dedicated DMA channels to be directed to peripherals. These channels allow specific memory-to-peripheral and peripheral-to-memory transfers to be made possible using DMA from peripheral memory that might otherwise not be addressable. This feature has the benefit of allowing multiple DMA transfers to occur simultaneously, however they must all be initiated by the CPU. Having the CPU initiate all partial transfers of images to the PL would not be ideal for a hardware accelerator because it would introduce additional latency to the system, creating initial limits on the total possible acceleration that could be achieved. In addition, the extra communication between the accelerator peripheral and the CPU would increase design complexity.

All interconnect information supported in:

<http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf>

The Accelerator Coherency Port (ACP) is a cache coherent interconnect between the PS and PL in the Zynq. For our project, this interconnect will not be ideal since cache values will be modified on memory access. This could result in cache thrashing and limit the performance of the OS and our image processing application, since the frequent large memory accesses made by the accelerator would be evicting data being used by other applications from the share cache. Since the data being accessed by the accelerator is not likely to be re-used by the CPU this will have a solely negative effect on system performance.

The AXI general purpose (GP) interconnects are useful for sending short messages between the PS and PL. This type of interconnect will allow PL access to the memory controller when configured through the central interconnect controller, but using this variety of interconnect is not ideal for large transfers since AXI GP interconnects lack FIFOs. In addition, these interconnects do not support a burst mode, can only handle up to 8 issued reads/writes, and are limited to a 32-bit data bus width. All of these factors make the AXI GP interconnects not a favorable option for image data transfer, however, they are effective, as previously mentioned, for short message passing. These interconnects will the appropriate choice for the peripheral control interface to the PS.

The AXI high performance (HP) interconnects provide a direct pathway from the programmable logic to the processing system memory controller, allowing the attached peripheral to initiate its own memory transfers. This interconnect features a configurable data bus width between 32 and 64 bits as well as a configurable burst length. These features allow the memory communication to be optimized for a given application. One of the most promising features of the AXI HP interconnects is the presence of FIFOs to buffer read/write values. These allow a large number of read and write requests to be placed and handled when there is an opportunity, lowering overall transfer latencies. Creating an accelerator with a master-mode AXI HP interconnect will allow the accelerator to initiate memory transfers whenever the peripheral requires additional image data, or must write new data to shared memory. This autonomous behavior will allow the issuing CPU to continue other operations or schedule new processes. A potential issue with using the AXI HP interconnect is the possibility of introducing too much memory traffic and slowing down CPU memory accesses. Peripherals will need to be carefully designed to prevent too much memory traffic.

# Current Implementation Plans

After performing significant research regarding image processing techniques and acceleration methods in hardware, we developed a tentative outline of our application implementation. Currently, we plan to capture image data from the USB camera using software functions available in the OpenCV library. After discussing this possibility with Avnet, they relayed to us that reading camera frames in hardware may have significant advantages based on throughput fps. Ultimately, we may choose to implement the hardware method if the reading method used in software inhibits our application from providing a quick response to user input.

Assuming we will capture image frames using software methods, frames will be stored in DDR memory using the PS. The process for utilizing the accelerators will likely take the following form. To begin an accelerated operation, an image frame will be transferred to a reserved frame buffer in DDR memory. The PS will then communicate to the PL using an AXI GP interconnect to configure the accelerator parameters for the operation. Next, the accelerator control register is set using a start command to begin the operation. At this point the accelerator will begin retrieving image data from DDR memory through an AXI HP interconnect. As the accelerator completes portions of its operation on the image, the results will be written back to a frame buffer in the DDR memory that holds the result of the operation. When the operation has finished, the accelerator will use an interrupt to notify the processing system that the manipulated frame may be read back from the result frame buffer.

Additionally, after researching the feasibility of several OpenCV image processing algorithms in hardware, we found several potential candidates that may be used for hardware acceleration. As a basic level, the color filtering methods used in many research examples appear to be excellent potential candidates for parallelism in the programmable logic. This is because we could effectively break each frame into parts and pass each part through a filtering algorithm in hardware simultaneously. As requested by Avnet, we also desired to implement more complex algorithms in hardware. One current possibility, as described in the object recognition section above, is contour tracing. Contour tracing is a likely candidate because we have found several research documents that perform this exact task and outline the high-level procedure for accomplishing this.

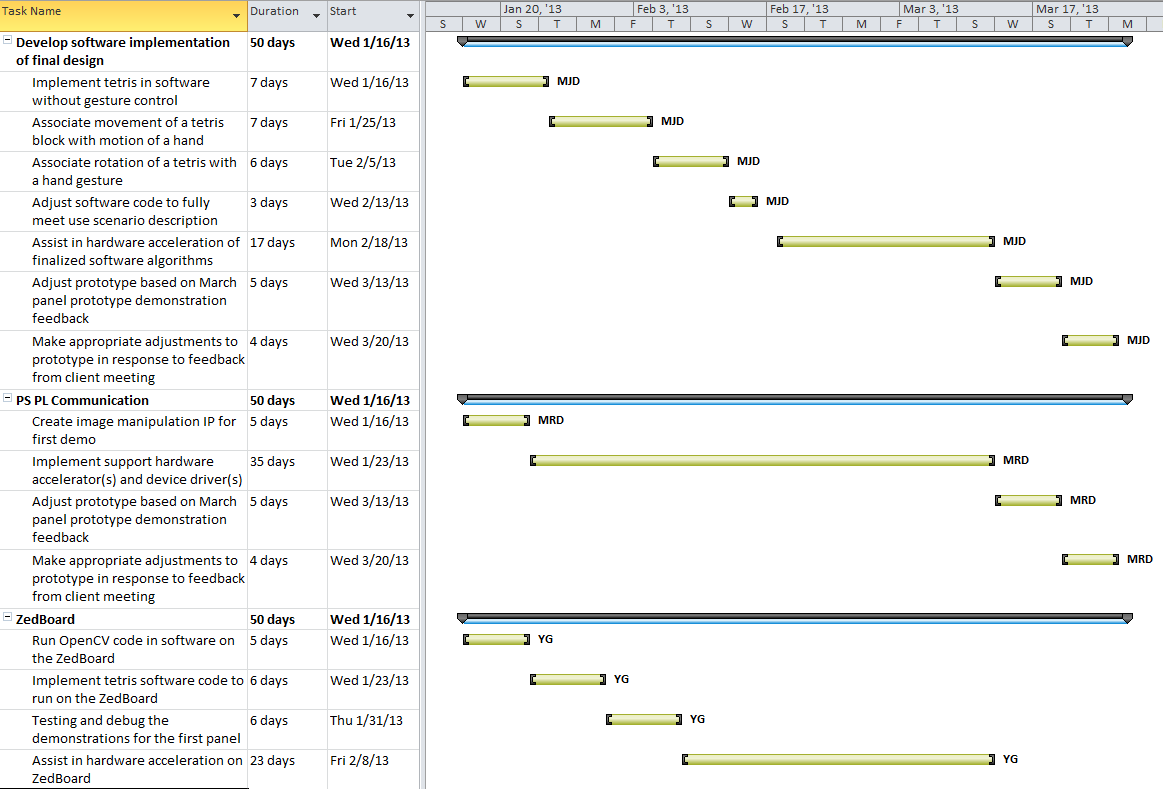
# **Application Performance Testing**

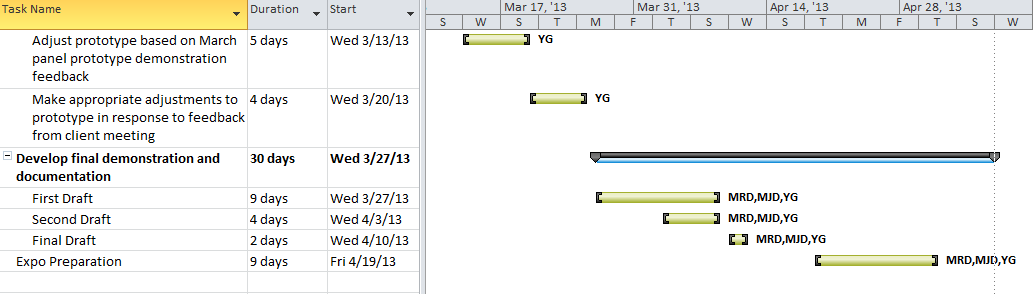
To provide a performance comparison of the software and hardware accelerated phases of the project at least two performance measurements have been identified. These performance measurements will be applied to both the software and hardware accelerated versions of the project for comparison.

The frames per second (FPS) throughput of the system for both the software and hardware accelerated versions of the project will be compared to assess the benefit of the hardware accelerator. To take this measurement the system’s image processing algorithm iteration duration will first be calculated by subtracting time of day measurements at the beginning and end of our algorithm. The reciprocal of this process duration will then be used as the system’s frames per second throughput. Successive FPS values will likely be averaged to lessen the effects of intermittent latencies introduced by the operating system.

The frames per second throughput of the image processing operation being accelerated will be compared to the software version of that operation. To calculate the time required to process a single frame, the the time of day will be recorded in code prior to beginning the operation. The time of day will then be captured again after the function call is completed. The difference of these two time values will provide microsecond precision for how long the operation takes to complete. The reciprocal of the operation duration will then be used to determine the frames per second throughput of the image processing operation. Successive FPS values will likely be averaged to lessen the effects of intermittent latencies introduced by the operating system.

# Tentative Schedule





## February Panel Demonstrations

1. Tetris game running on ZedBoard

As the first panel of next semester, we expect to have a completed version of the tetris game running of the ZedBoard. We expect to use the VGA output of the ZedBoard to display the game screen on an external monitor. The operation of the tetris game will reflect standard game rules, as outlined in the use scenario.

2. Implemented controls for movement of a tetris block using image processing

With the tetris game, we also expect to be able to fully control the horizontal location of a falling tetris block by using image processing on the ZedBoard to detect the motion of the user’s hand. The behavior of the tetris piece should follow the guidelines outlined by the use scenario.

3. Implementation of an image intensity inverted using hardware on the ZedBoard

We also plan to demonstrate an example of hardware acceleration on the ZedBoard by passing an image through an image intensity inverter in the programmable logic.