# Digital System Design Proposal: Video-Painting

*Opal Densmore (997273604)*
*Kei-Ming Kwong (997595026)*
*Wahid Rahman (997443770)*

Digital System Design (ECE532H1S)
TA: Jasmina Vasiljevic
February 12, 2014

# 1.0 Table of Contents

# 2.0 Project Information

## 2.1 Introduction

The use of video processing in human-computer interaction (HCI) is expanding in applications ranging from entertainment, health-care, and education. Gaming consoles like the Xbox and its Kinect camera sensor augment users' multimedia experience: computer vision and image processing algorithms built around the Kinect has been used to control in-game avatars based on user facial expressions and biometrics [1]. A biomedical company, Gestsure, is developing a Kinect-based system that uses surgeons' hand motions to retrieve patient scans and images during operation [2]. In addition, studies note that the "customized software to facilitate classroom interactions and to create Kinect-enabled contents seems to be missing in the picture of current technology integration" [3].

The goal of this project is to expand this video-based HCI to a proof-of-concept design on an FPGA. In particular, this project implements a video painting application that runs on video processing hardware in the FPGA. The motivation is to demonstrate real-time video HCI on an embedded FPGA system. Some applications of this project are as listed below:

- Interactive video, for a growing field for entertainment uses like the Kinect camera.
- Teaching, particularly for presentations conveyed over the Internet.
- Art, where our video drawing system could form a new creative medium.

### Prior Work

While several video-based HCI systems have been developed as discussed previously, two systems implement a video painting system similar to this project.

"Virtual Whiteboard" [4]

This system develops an electronic chalkboard for teaching, without using specialized hardware. Instead, it uses generic camera and projectors. This software system uses Java and OpenCV libraries to detect and predict hand motions in real-time video. This system is a robust and feature-rich solution: it recognizes hand gestures by both hands at a real-time video frame rate (30 frames/sec at 320x240 resolution). However, the system only produces a screen of drawn shapes, not real-time video.

cvPaint - Simple drawing using camera [5]

This system is an OpenCV project that captures video from a webcam, tracks a user pointer (e.g. yellow block), and paint on the output streaming video based on location of pointer. This project serves both as an inspiration and an example for this project. It is a simple algorithm implemented using the high-level OpenCV API. One downside is it relies on an OpenCV-compatible operating system. The reliance on software means limits the system's frame rate.

Why Use FPGAs?

The benefits of using FPGAs for this project are twofold: hardware acceleration and reconfigurability. By implementing this project on an FPGA, custom hardware cores can be developed to accelerate video processing. This can directly improve frame rate. Reconfigurability allows for fast design cycles: redefining and testing software and hardware can be done relatively quickly using an FPGA.

## 2.2 Goal

Create a new form of media which tracks user input to create live content on a high quality real-time video stream.

## 2.3 Team

### Kei-Ming Kwong

Kei-Ming is experienced with programming hardware systems and software development from his internship at Altera Toronto and his thesis project (C/C++, Perl, Tcl, Python). He is also experienced working with Verilog from his summer research. He has limited simulator experience with complicated designs and systems.

### Opal Densmore

Opal has experience with software from her PEY internship and course work (C/C++, Perl, Python). She has experience with Verilog from a summer research internship and from her fourth year thesis project. Opal has limited simulator experiences with complicated designs and systems.

### Wahid Rahman

Wahid has experience in FPGA video processing from previous summer research. He held an internship at Altera Toronto developing a Verilog IP core for use in Quartus and simulation in ModelSim. He has some software experience in C/C++ and Perl.

# 3.0 System Overview

## 3.1 System Description

The design will make use of a Xilinx MicroBlaze soft processor to implement the Video-Painting algorithm. In order to implement a real-time system, critical software functions will be hardware accelerated. The system will integrate these hardware IP with the MicroBlaze to process real-time Video-Painting.

## 3.2 Basic Features

### Video Painting Modes

1. Free Draw Mode → superimposes the drawing with live camera feed.
2. Still Draw Mode → superimposes drawing with still picture.

### Palette Control

1. Brush Colour Selection → DIP Switch control of RGB values.
2. Brush Control → Width/Type/Erase.

## 3.3 Key Definitions

**Draw Frame**: An intermediate frame which is used to store the user drawn data.
**Video Frame**: An input frame from the camera which will be used as a background
**Composite Frame**: an output frame with the Draw Frame overlaid on top of the Video Frame

## 3.4 System Flow

The system flow diagram describes the generic flow of the algorithm with broken down into modular components. It highlights the different connections between functions of the algorithm and memory.

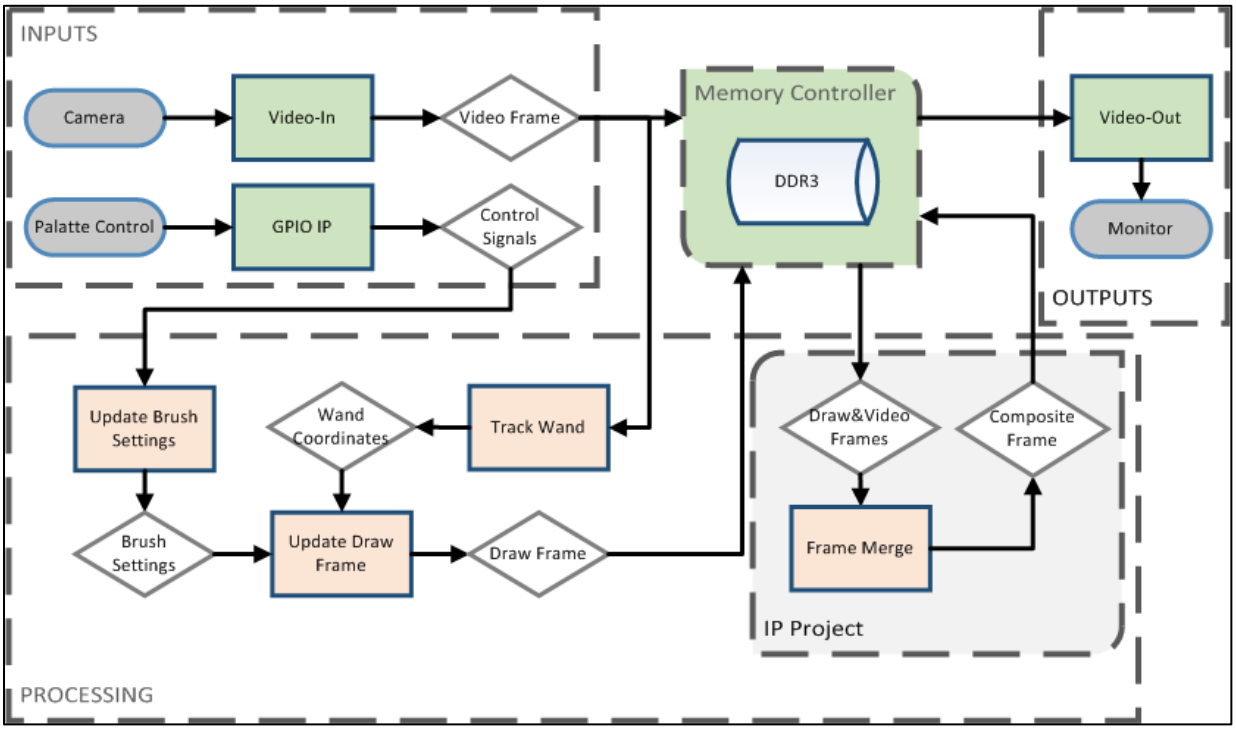


**Figure 1** - Overall system flow diagram. Green blocks show existing blocks, red blocks show blocks that need to be created. Diamonds represent intermediate data.

## 3.5 System Architecture Diagram

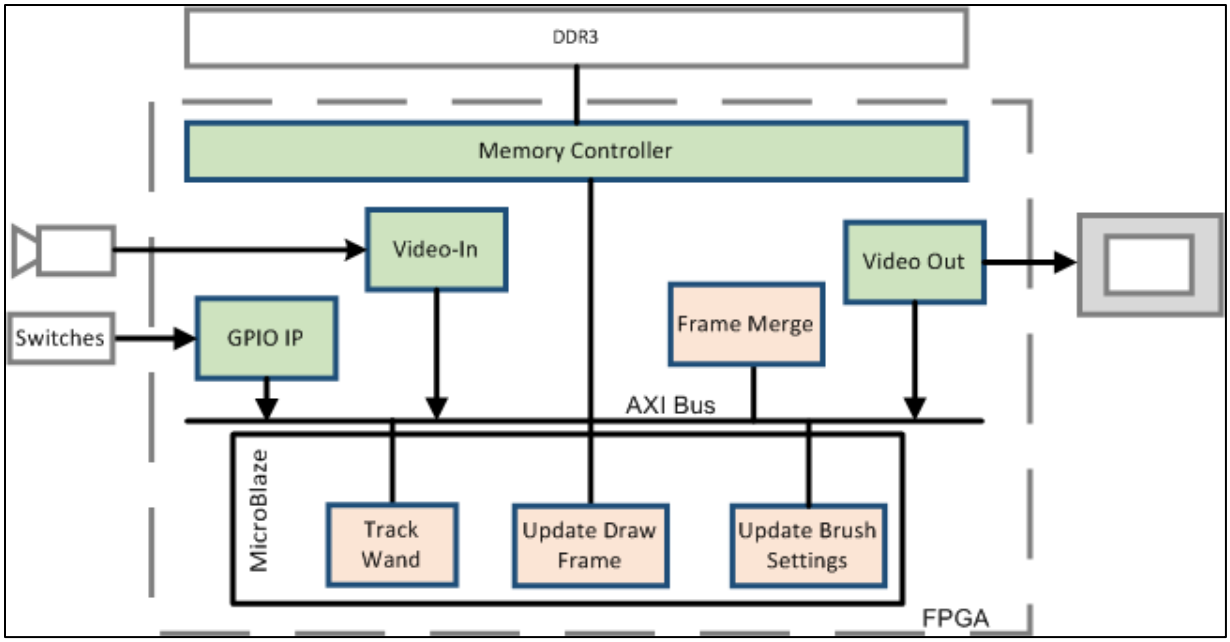


**Figure 2** - System architecture. Green blocks show existing blocks, red blocks show blocks that need to be created. Blocks within the MicroBlaze are implemented as software.

### 3.6 Functional Block Descriptions

The following describe the functional blocks which will need to be developed.

#### Track Wand (Software)

The Track Wand block will monitor input video frames, locate an LED of a certain colour (hereinafter referred to as "Paint Wand") and output its pixel coordinates within the frame.

**Inputs**: The video frame as made available by the HDMI video-in IP.

**Outputs**: The horizontal and vertical pixel positions of the LED in the input frame.

#### Update Brush Settings (Software)

The Update Brush Settings block takes user input from the GPIO switches on the Atlys board and translates them into brush settings. Valid settings include brush colour, brush width, and brush type.

**Inputs**: Switch values taken from GPIO IP.

**Outputs**: Brush setting values.

#### Update Draw Frame (Software)

The Update Draw Frame block uses brush location, brush settings and Draw Frame address to update the Draw Frame. The brush settings will be used to determine a stencil of the brush, which will be written to the Draw Frame at the specific brush location.

**Inputs**: Brush settings, Paint Wand coordinates, draw frame memory address.

**Outputs**: Updated Draw Frame, written directly to DDR3 memory.

#### Frame Merge (Hardware)

The Frame Merge block reads both the Draw Frame and Video Frame from memory, produces the Composite Frame based on the Video and Draw Frames, and stores the Composite Frame in memory.

**Inputs**: The memory addresses corresponding to the Draw, Video, and Composite Frames in DDR3 memory.

**Outputs**: Composite Frame, written directly to DDR3 memory.

## 4.0 Specifications

### 4.1 Constraints

The following is a list of constraints in the standard use scenario of our system

- Fixed distance between artist and camera.
- Use of a solid R/G/B Paint Wand calibrated to the system
- No similar colours in scene to Paint Wand

### 4.2 Resource Requirements

- Camera
- Pure RGB light source, i.e. the Paint Wand
- Atlys Board
- HDMI Monitor

# 5.0 Design Approach

## 5.1 Design Stages

### Software IP Development

During the initial stage of development, the goal is to implement a C program to refine the painting algorithm prior to hardware implementation. The C program will implement the system fully in software on the MicroBlaze processor.

### Hardware IP Development

During this next stage of development, the goal is to accelerate critical software functions in hardware to improve performance. In this stage, the following will be done:

- Implement hardware IP for critical system functionality (cf. Fig. 2).
- Develop software control interface for hardware IP.

### Integration

During this stage of development, the goal is to implement the full system using the software and hardware components. The final product will be a demonstration of the Video-Painting system. This stage can be divided into the following substages:

- Integrate all pre-existing blocks with components of the software program to create a live video stream.
- Enable dynamic tracking and update of Draw Frame.
- Integrate the Frame Merge hardware IP into the system with video frames.

## 5.2 Testing

### Hardware IP Testing

The testing procedure will involve the following tasks:

- Test functionality of custom hardware IP blocks using ModelSim.
- Verify functionality of pre-existing blocks, such as Video-In/Video-Out.
- Verify software control interface for each hardware IP independently.

### Integration Testing

The testing procedure will involve the following tasks:

- Verify working live video stream.
- Benchmark performance of video throughput.
- Verify bus protocol (AXI) works for data transfer between software, hardware IP, and DDR3 memory.

## 5.3 Risk and Contingency Plan

### Risk Factor

Due to the composition of complex hardware and software components, the following are areas of risk:

- Interface issues with AXI protocol and custom IP
- Real-time video performance (bottleneck in memory accesses, etc)

### Contingency Plan

Our contingency plan identifies some of the issues we may encounter while working on this project.

#### Performance Issues

One bottleneck we may encounter is that one of the algorithms implemented in software is too slow for our needs. In this case, we would accelerate portions of the algorithm with a hardware block.

#### Bus Protocol Issues

If the AXI bus protocol becomes a bottleneck, due to either performance or design complexity, we will implement direct data transfer between IP blocks instead.

#### Memory Issues

If there is an issue with memory access speed, we will move critical sections of the memory such as portions of the Draw Frame to on-chip memory in order to reduce this latency.

#### Software Alternative

Another possible issue is that it will take too long to implement the Frame Merge IP. In this case, we would use our software version of this IP instead. This would guarantee correctness, however, our design will likely suffer due to slower algorithm.

## 6.0 Milestones

The overall goal of this project is to complete a fully integrated and tested demonstration by Apr. 2, 2014. To achieve this, we plan to complete integration by Week 5 (Mar. 26, 2014). This allows overhead for system-level debugging and design revisions, if required. To achieve this high-level goal, the following schedule has been developed:

#### Week 1 (Feb. 26, 2014): Develop software IP

**Kei-Ming**: Finish of software Track Wand IP with static inputs.

**Wahid**: Development hardware core for Frame Merge IP for use with static frames.

**Opal**: Initial development software Update Draw Frame and Update Brush Settings IPs independently with static inputs.

#### Week 2 (Mar. 6, 2014): Finalized software algorithm flow and start hardware testing

**Kei-Ming**: Verify functionality of pre-existing Video-In IP.

**Wahid**: Develop AXI bus interface control for Frame Merge IP.

**Opal**: Complete software Update Draw Frame and Update Brush Settings IPs independently with static inputs.

#### Week 3 (Mar. 12, 2014): Individual Hardware IP Testing

**Kei-Ming**: Continue development in Video-In IP and test functionality.

**Wahid:** Develop software control interface of Frame Merge IP.

**Opal:** Integrate software IPs Track Wand, Update Draw Frame, Update Brush Settings, and the software version of the Frame Merge IP using static inputs for all IPs. Note Update Brush Settings should use dynamic inputs.

Week 4 (Mar. 19, 2014): Start of Integration Stage

**Kei-Ming**: Finalize software interface for Video-In IP

**Wahid**: Test Frame Merge IP with software control interface.

**Opal**: Integrate software with Video-Out IP.

Week 5 (Mar. 26, 2014: Preliminary integration done

**Team**: Integrate Frame Merge IP for use with dynamic frames (e.g. input video frames).
Integrate software IPs Track Wand and Update Draw Frame with dynamic frames (e.g. input video frames)

Week 6 (Apr. 2, 2014): Final Test

**Team**: Integrate final system with dynamically changing Draw Frame (e.g. tracks user input)

Week 7 (Apr. 9, 2014) Project Demo

**Team**: Everything working as specified in this document.

# 7.0 References:

[1] Z. Zeng, “Microsoft Kinect Sensor and Its Effect,” *IEEE Multimedia*, vol. 19, no. 2, pp. 4-10, 2012.

[2] M. Campbell, “Kinect imaging lets surgeons keep their focus,” *New Scientist*, vol. 214, no. 2865, p. 19, 2012.

[3] H. J. Hsu, “The Potential of Kinect in Education,” *International Journal of Information and Education Technology*, vol. 1, no. 5, pp. 365-370, 2011.

[4] F. Tavakolizadeh. (2014, Jan. 10). *cvPaint - Simple drawing using camera* [Online]. Available: http://farshid.ws/projects.php?id=112

[5] M. Lech, B. Kostek, A. Czyzewski. “Virtual Whiteboard: A gesture-controlled pen-free tool emulating school whiteboard,” Journal of Intelligent Decision Technologies, vol. 6, no. 2, pp. 161-169, 2012.