# EECS 443 Final Project: Etch-a-Sketch™ Clone

## Team Members & Participation

Our team consisted of Bradley Gibbons and Stephen Longofono. We each contributed in equal parts to the design process. The individual source files have the original author’s name, but we both worked on most of the files.

## Description of Project

## The project implements a clone of the children’s toy “Etch-a-Sketch™.” The toy features a screen which can be drawn on by rotating two knobs, each of which moves a cursor in the vertical and horizontal directions respectively.

## Our implementation consists of a “canvas” on the VGA monitor, and the directional buttons of the Nexys4 are used to draw lines on the canvas. Twelve switches are used to change the color of the line being drawn, and the center directional button is used to enable or disable drawing. This allows the cursor to be moved around the screen without drawing, so that more interesting pictures can be drawn.

We had intended to use the accelerometer to reset the screen, but we ran into issues with the way Vivado synthesizes block RAM (more on this below in the design section). Instead, when the board is shaken, the screen resets, but the picture is preserved. We propose a possible solution in the future work section.

## Description of Modules and High-Level Design

The high level design consists of six parts: a driver module, an accelerometer module, a button controller module, button debouncer modules, a VGA controller module, and a memory module. The driver stitches together the other pieces and maps their inputs and outputs to appropriate sensors and actuators on the Nexys4 board. The button controller handles the button inputs and tracks the cursor position on the virtual canvas. The accelerometer unit determines when the board is being shaken. The VGA controller drives the VGA monitor and interfaces the memory unit. The memory unit stores color values for each of the 307200 pixels which represent the virtual canvas.

### Module Design: Driver Unit

The driver unit is the top level module, used primarily to link constituent parts together. For inputs, it includes 12 switches indicating the color values to be passed to the VGA controller, 1 switch for system reset, and one signal from the SPI interface for duplex communication. For outputs, it includes signals mapped to the VGA controller, the accelerometer module, and the button controller.

### Module Design: VGA Controller Unit

The VGA control defines the interface to the VGA monitor hardware, and tracks the state of a 2-D array (Block RAM) representing the pixels on the canvas. Each array element is a 12-bit vector representing the color of the pixel. The interface follows the VGA specifications and timing per the standards (found online). Its inputs include logical signals representing what pixel to write and what color to write, along with an enable/disable bit for writing to the BRAM. Its outputs include signals to the onboard DAC, which directly drive the VGA hardware.

### Module Design: Accelerometer Unit

The accelerometer unit is responsible for determining when the board is shaking. It uses the Xilinx ADXL362Ctl module to read from the on-board sensor over a SPI interface module (both written by Xilinx developers). The highest magnitude in each of three directions is read, and the logical OR of these signals makes up the output signal. For inputs, it includes the system clock and reset. Its output is a logical signal indicating whether or not the board is being shaken.

We had intended the signal to be used to reset the canvas to blank, but we ran into issues with the way block RAM is synthesized in Vivado. Apparently, there is no way to reset a block RAM by a single VHDL command; every entry must be explicitly overwritten.

### Module Design: Button Controller Unit

The button controller is responsible for keeping track of where the cursor is, and moving it according to input from the five button debouncer units. The coordinates are translated into an address into memory, which is passed to the VGA controller to determine where to write. One of the buttons is a write enable, which is passed directly to the VGA controller to enable or disable writing to the canvas memory. Inputs include logical signals representing what pixel to write and what color to write. The outputs are signals to the onboard DAC, which directly drives the VGA hardware.

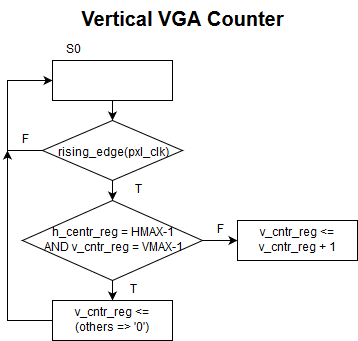
### Module Design: Button Debouncing Unit

The button debouncer unit is designed to filter out extraneous button presses (debouncing). Each has an internal counter which is used whenever the input signal changes. If after 10ms the signal has the same value, that value is output. Otherwise, the output remains as it was. Its inputs are the signals from the physical buttons on the Nexys4 board. Its outputs include debounced signals, that is, signals that have had the same value for 10 ms.

### Module Design: Memory Unit

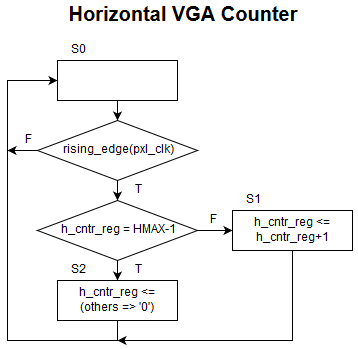
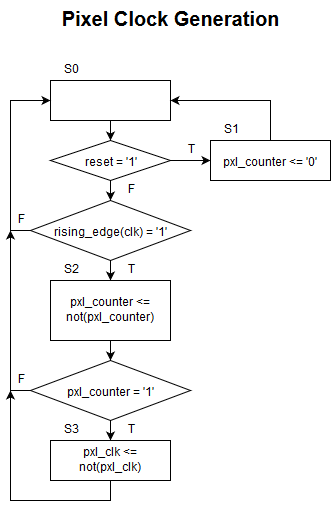
The VGA memory defines a simple write before read style RAM. If the write bit is high, the write data is written to the internal array at the write address passed in. The read data is populated with the data at the read address concurrently. Inputs include integer addresses into the internal memory array, a vector representing the data to be written, and a bit to enable/disable writing. The output is a vector representing the data read from the internal array at the read address passed in.

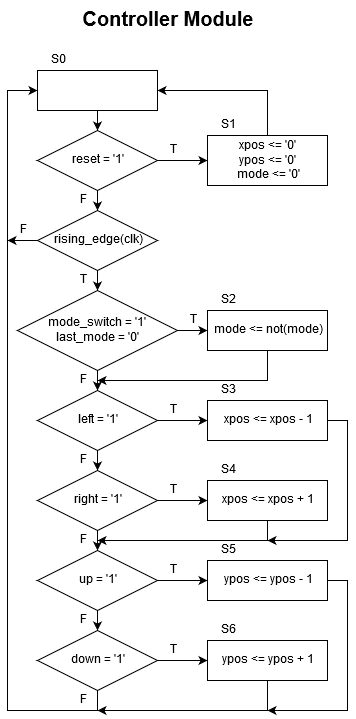
## ASM Charts for Key Modules



S2

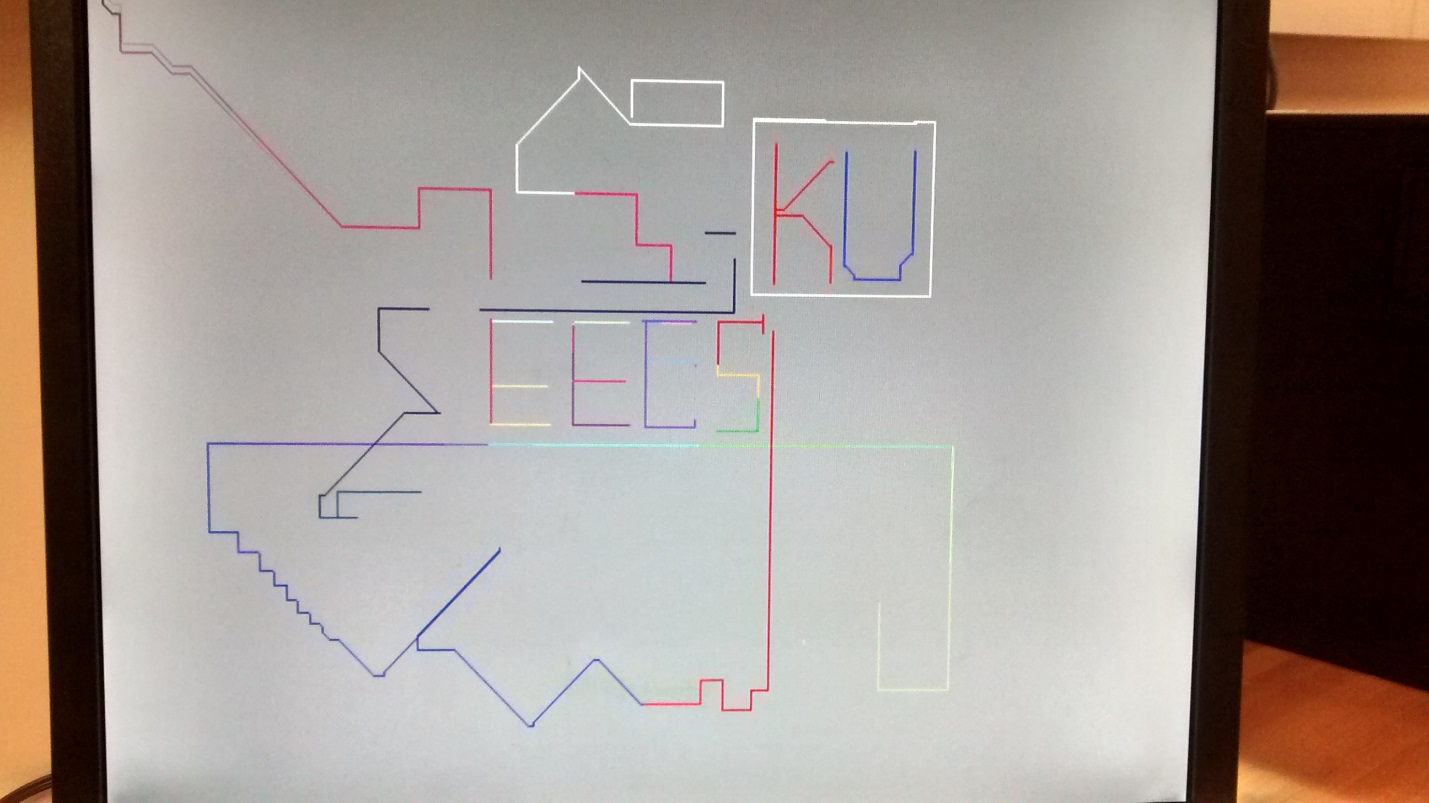
S1





## Pictures & Multimedia

The project demonstration:



Video demonstration:

<https://youtu.be/eSbEV-jCAHA>

## Future Work

There is a much room for improvement on our project, and it could easily serve as a base for a more advanced project. We essentially developed an interactive VGA platform, on top of which one could build a game, a display GUI, or any other project with a need for an interactive display. Our specific ideas for future work are as follows:

* Reset module – To overcome the shortcomings of the Vivado BRAM synthesis, a resetter module could be developed to zero out all memory locations. This unit would be triggered by the accelerometer unit, and its interface into the memory would have to be multiplexed with the VGA controller’s interface to avoid conflicts.
* Joystick operation – The button inputs are awkward for drawing with, and would be much easier to use with a joystick as input. This would involve using the onboard GPIO to map the button inputs to the buttons within a joystick sensor, but otherwise would not require much modification.
* Wacom operation – For even more sophisticated drawing, a Wacom tablet and capacitive touch stylus could be used to directly specify the pixels to be written. This would require many changes to our existing code, as the pixel to be written is decided far upstream from the physical inputs.
* Stroke Width – At present, the cursor draws lines which are a single pixel wide. With a few modifications, we could use the leftover switches to control the width of the line. According to the width set, the memory would write the pixel passed in along with a number of adjacent pixels within a certain radius.