# BASTARD ICE CREAM

## PROJECT DESIGN

## EMBEDDED SYSTEM (CSEE 4840)

## PROF: STEPHEN A. EDWARDS


HAODAN HUANG

DEPARTMENT OF ELECTRICAL ENGINEERING

hah2128@columbia.edu

LEI MAO

DEPARTMENT OF ELECTRICAL ENGINEERING

lm2833@columbia.edu

ZIHENG ZHOU

DEPARTMENT OF ELECTRICAL ENGINEERING

zz2222@columbia.edu

YAOZHONG SONG

DEPARTMENT OF ELECTRICAL ENGINEERING

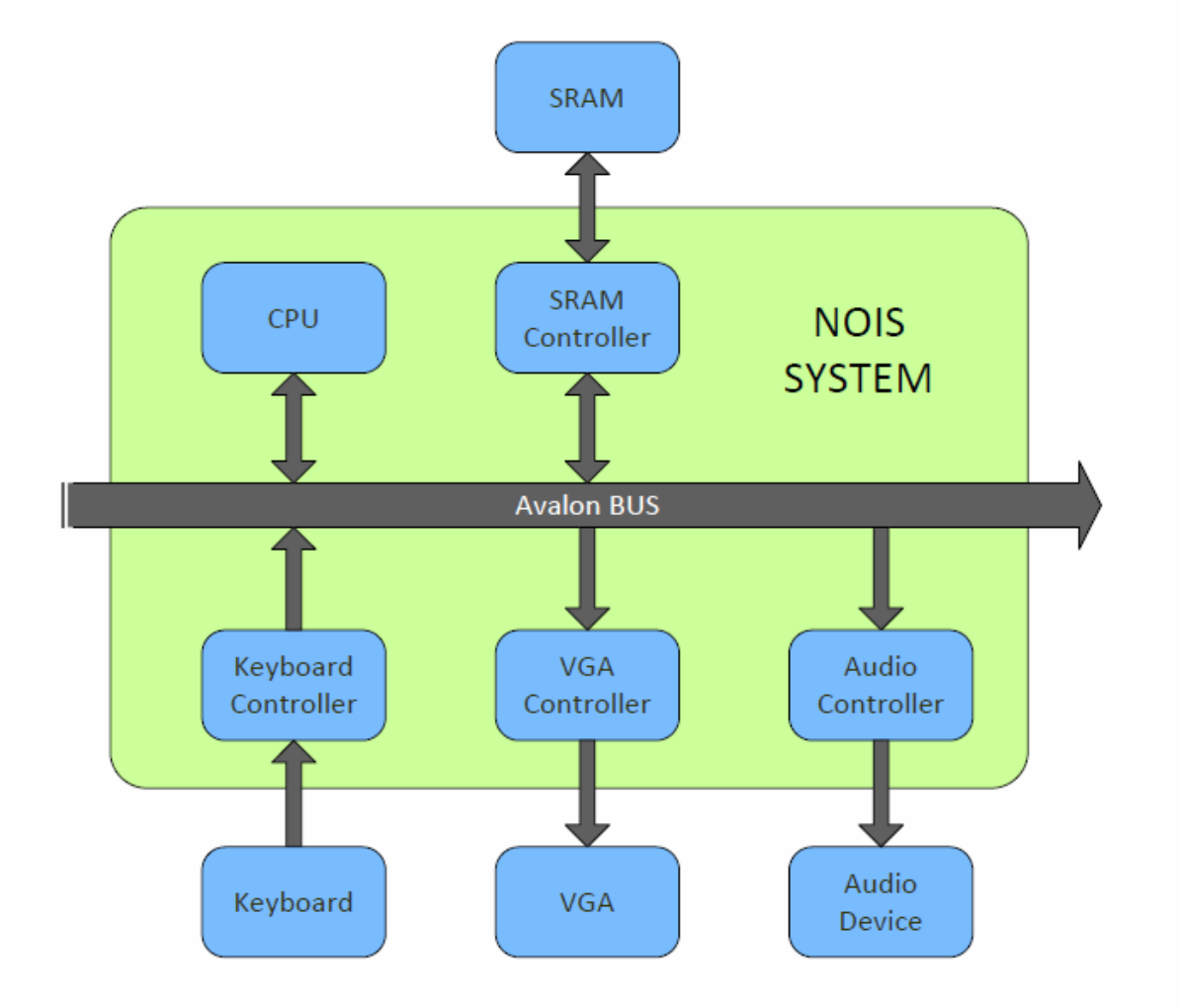
ys2589@columbia.edu

## I. INTRODUCTION

In this project, we intended to implement a simple video game called “Bastard Ice Cream”. The general purpose of the game is to eat up all the fruit while avoid colliding with bastard enemy. The player is able to create and destroy ice cubes which could block the enemy or escape. The player could only win the game if he eats up all the fruit in the specified time interval. The game will ends if the Ice Cream is capatured or time runs out.

## II. ARCHITECTURE

Based on the experience of lab 3, in this project, we will also build a nios system component which contains several sub-components like VGA Controller, CPU, SRAM. The peripherals connected with this system are Keyboard and Audio Devices.



## III. HARDWARE IMPLEMENTATION

Hardware is responsible for sprites' rendering and audio synthesize. We will store lots of sprite arrays in the hardware and render these patterns on screen by control array. We plan to make each sprite size as 32x32 pixels and the whole screen is 640x 480. So the control array should at least 20x15. However, considering the development of our project, we make the control array as 30x20 which has a map larger than the screen.For character rendering, we will make its coordination as 640x480. It means the "Ice Cream" will not walk one tile by one tile, but one pixel by one pixel.

## IV. SOFTWARE IMPLEMENTATION

### 1. Control of the movement of the "Ice Cream"

"Ice Cream" is the character controlled by the player via keyboard, specifically through up/down/left/right arrows, thus, we need four different angles to represent the state of its movement; the respective graphics are as follows. (Note that this is only an example/prototype, the final representation might be different from these according to design modification.)


As the player inputs direction queries by pressing arrow buttons, the C program will read in these command and use switch sentences to determine which reaction to be executed. Another fact needs to be noticed is before every movement of the character, the program should check first if the tile ahead has been "blocked" (here, block means the tile originally has a ice cube or has been later frozen).

### 2. Design of the "Guard"

The "Guard" is the character walking around the map and somewhat becomes the enemy of "Ice Cream". In first several maps there will be only one "Guard" each game and the "Guard" just "randomly" wanders around, here, we will use a algorithm to regulate the movement of

“Guard” rather than random; As the game goes on there would be more “Guards” and it will periodically chase the “Ice Cream” other than wandering around. Therefore, we need another timer to record the period of chasing and also, an algorithm to follow the track made by the player’s character. Of course, the movements of the “Guards” also require pre-check with the in front tile availability.

3. The animation of the Ice cube generation and destruction

There would be a process of the generation of a line of ice cubes when the player presses space bar, shown as follows.

Generation

Destruction

4. The encounter of the “Ice Cream”, Fruit and the “Guards”

When the “Ice Cream” walks through the fruit, the fruit will disappear and thus been eaten by the “Ice Cream”, in the meantime, there should be a counter counting the numbers of the fruits eaten by it, if the number reaches the original 3 number of fruits (or we may set the number of fruits as constant, e.g. 15).

When the “Ice Cream” meets the “Guard”, it dies and game over. Therefore, we intend to label each 32x32 tile with an coordinate position with respect to a “original” tile, thus, to determine the encounter of two objects, we simply compare the coordinates of the two objects, e.g. if (x1-x2)==0 && (y1-y2)==0, return 1.

## V. VIDEO

VGA Control

Character design, monster design, these stuff we already learned from lab3, we set this in the RAM and connects to nios2 to control the position of the character. Based on 32-32 pixel, our screen can be divided into 20-15 pieces. The animation of the attack should be done in software. Each 32x32 pixels array on the screen will be represented by two strings of nine hexadecimal values. Each group of nine values will represent a mapping in one color(red,blue ,green).With combination of three strings, we can create each array with different images.

Basically , we have two sprites in the game. The bastard Ice Cream and the Guard. Each contains four positions for motion : “UP”,”DOWN”,”LEFT”,”RIGHT”. We will continue to use the 3 strings to create the image for both sprites. Each one consists of 32-32 pixels.

Animation control(time delay): when the Ice Cream generate the ice cubes or destroy them, it could not move for a certain period of time.

For the ice cubes to form, we count that it travels across each array for 0.5 seconds. If the sprites movement is slower than this, then we treat the array as unavailable.

## VI. AUDIO

The basic FM equation is

$$x(t) = \sin(\omega_c t + I \sin(\omega_m t))$$

where x(t) is the amplitude at time t, ωc is the carrier frequency (the fundamental tone we hear), ωm is the modulating frequency, and I is the modulation depth. The timbre of the sound is largely determined by the ratio ωc/ωm, which is generally set to an integer ratio (e.g., ωc = 3ωm).

The fundamental frequency of musical notes follow an exponential scale. The A above middle C is 440 Hz, and going up an octave doubles the frequency.Western music is built on a scale of twelve semitones, each in equal ratio. Thus, the frequencies of a standard scale are of the form

$$f = 440 \cdot 2^{p/12}$$

where f is the frequency in Hertz, p = 0 is the A above middle C, p=1 is A♯ , p=2 is B ,p=3 is C, p=12 is the A the octave above, p = −12 is the A the octave below, etc. Each sound will be trigger and terminate by software. Based on these basic sound, we could produce several music: Game start, Game Over, Generate & destroy ice cubes, Collect the fruits.