Card Counting
Project Design

Team Members
Christos Savvopoulos
Hugh Gordon
Nathan Rogan
Card Counter

Overview
Our goal in this project is to take real-time data from cards and interpret that into a future probability of the cards to come. Using a camera as input to the system, the digital data retrieved will be allocated into registers based on color levels determining a unique identifying pattern for each card. This process will allow a characteristic mark for the cards and a basis on which to determine what a future card is classified as.

After the system is loaded we will start a statistical summary of cards viewed and adjust the probabilities of future cards in real time. This section will be done in software.

Block Diagram
**Components**

**Camera:**
- input: x,y: integers
- output: r,g,b: 8bits each
- on every clock tick (r,g,b) is the value of the color at position (x,y) on the camera.
- The range of x,y is implementation dependent.
- If the input is out of range r,g,b is undefined.

**Sync:**
- output: reset(bit), x,y(integers)
- This module first sets reset=1. Then, on the next tick, it sets reset=0 and iterates x and y through their range. This is done by iterating first over y and then over x, e.g.:
  - for (y=0; y<480; y++)
  - for (x=0; x<640; x++)
- [Note: the code is only for illustration purposes, sync is a hardware module]
- When it has iterated over all values it starts over (reset=1 and so on).

**Conv:**
- input: r,g,b: 8bits each
- output: c: 2bits, representing either of red, black, white, green
- conv converts the RGB value to a color that is one of the possible colors on a card. It does so by using thresholds.

**Decoder:**
- standard 2-bit decoder
- counters: standard up counters with reset signals
- The counters count the number of pixels per color. In essence the hardware implementation produces a histogram of the visual input. The decoder is responsible for selecting a counter.

**Interface:**
- input: four integers
- The software will poll the hardware implementation from time to time to ask for the values of the four counters. The interface is responsible for storing the values of each counter right before a reset signal is produced by the sync module. The interface also includes the bus interface, responsible for communicating with the software.

**Software:**
- The software is open ended. Our plan is to make it so that it first reads and "learns" the values of each of the cards' histograms so that it can recognize them without assistance after the learning process.
Pseudo C program to interface with hardware.

// predefined constants
#define NEW_CARD 10
#define MATCH_CARD 01
#define DO NOTHING 00
#define RESET 11

// data type
struct _histogram
{
    int red;
    int green;
    int blue;
    int black;
    int white;
    int yellow;
};

struct _card
{
    _histogram hist;
    char* card; // string card name
    int value; // unique id
} card;

// definitions
bool doMatch(_histogram c, _card cs[]); // match card c to deck cs
void saveCard(_histogram c, int value, card cards[]); // add card
void doReset(); // clear the data
void calcStats(); // future implementation of card stats
void listener(); // poll to catch and handle events
void listener(); // poll to catch and handle events

bool equal(_histogram s, _histogram m);
_histogram readRegStates();

_card cards[64]; // hold new cards
bool event_raise; // hardware event
int event_cond; // hardware call

int _tmain(int argc, _TCHAR* argv[])
{
    listener();
    return 0;
}

bool equal(_histogram s, _histogram m)
{
    return (s.black == m.black && s.blue == m.blue && s.green == m.green && s.red == m.red && s.white == m.white && s.yellow == m.yellow);
void listener()
{

    bool eventSaveCard=false;
    //poll catch events
    for(;;)
    {

        if(event_raise)//event rise
        {
            //switch on hardware call
            switch(event_cond)
            {
                case NEW_CARD:
                    eventSaveCard=true;
                    break;
                case MATCH_CARD:
                    if(eventSaveCard)
                    {
                        _histogram hs= readRegStates();
                        saveCard(hs);
                        eventSaveCard=false;
                    }
                    else{
                        histogram hs=readRegStates();
                        doMatch(hs,cards[] cs);
                    }
                    break;
                case DO NOTHING:
                    break;
                case RESET:
                    cards[]=null;
                    break;
            }
        }
    }
}