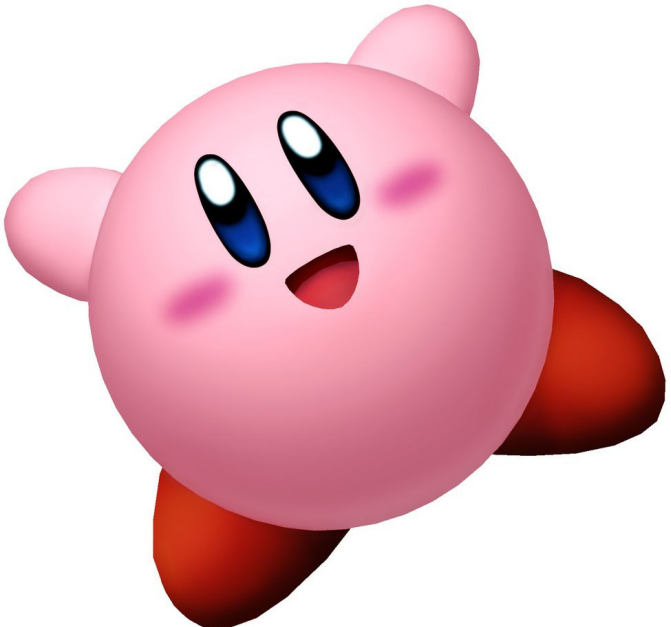
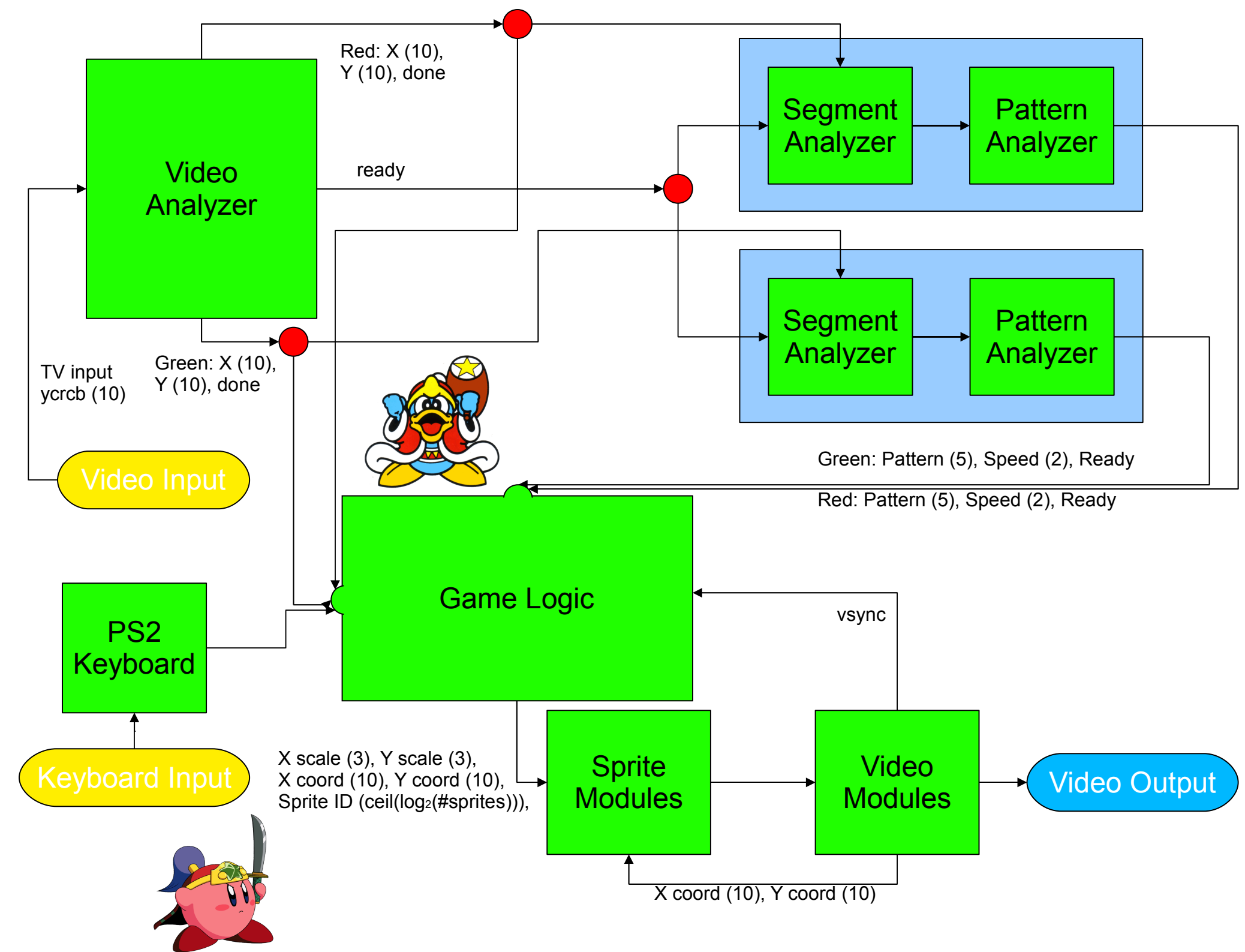
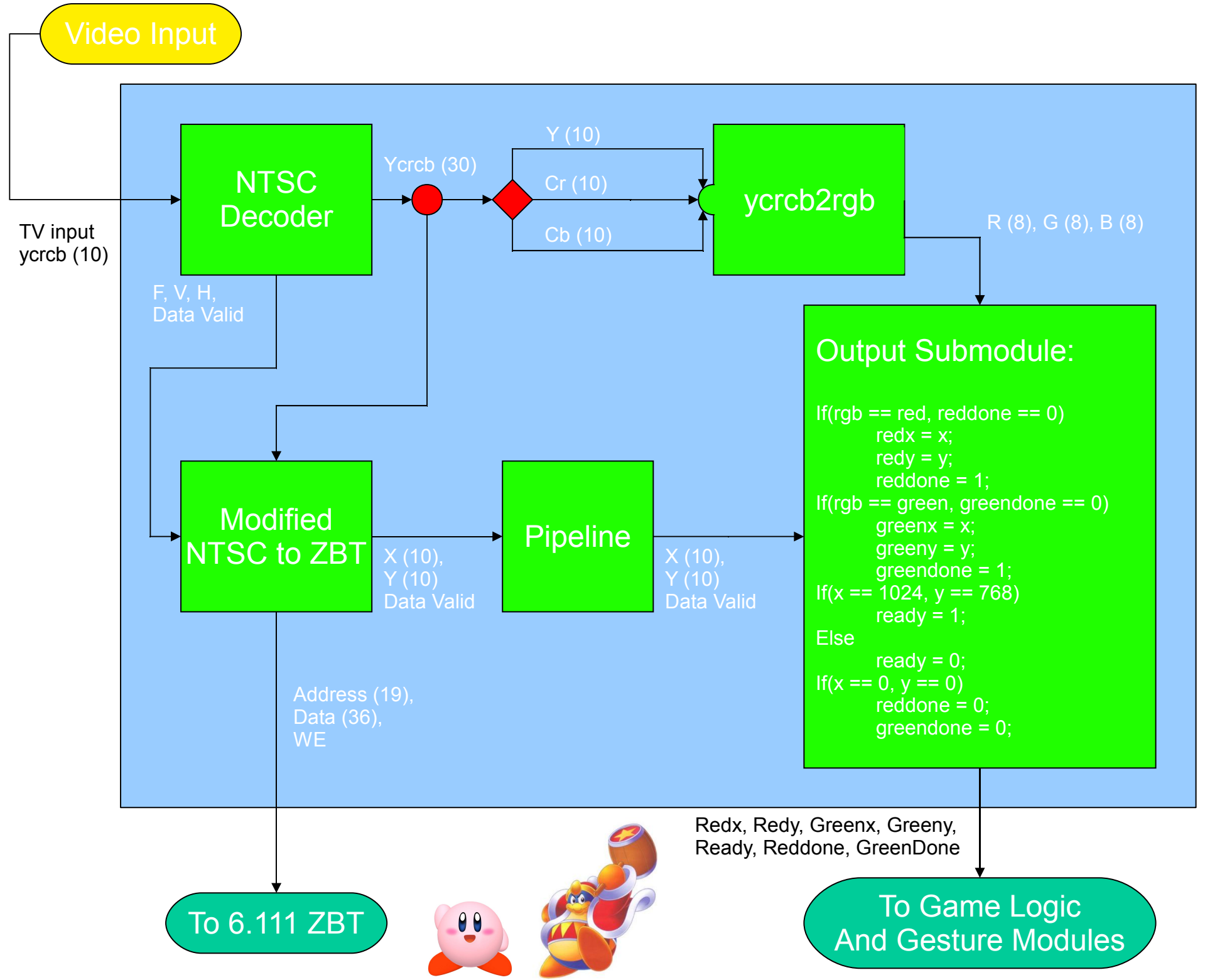
Kirby
vs
King Dedede
(super ultra laser attack game)



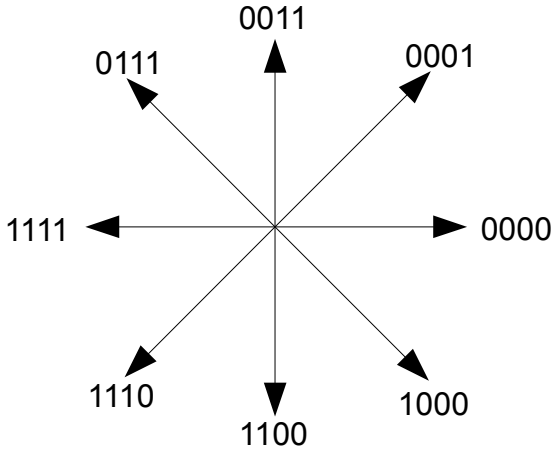
Daniel Gerber
Tynan Smith


Video Analyzer
Red: X (10), Y (10), done
ready
Green: X (10), Y (10), done
TV input ycrcb (10)
Video Input
Segment Analyzer
Pattern Analyzer
Segment Analyzer
Pattern Analyzer
Green: Pattern (5), Speed (2), Ready
Red: Pattern (5), Speed (2), Ready
Game Logic
PS2 Keyboard
Keyboard Input
vsync
X scale (3), Y scale (3), X coord (10), Y coord (10), Sprite ID (ceil(log2(#sprites))),
Sprite Modules
Video Modules
Video Output
X coord (10), Y coord (10)


Video Input
TV input
ycrcb (10)
NTSC
Decoder
Ycrcb (30)
Y (10)
Cr (10)
Cb (10)
ycrcb2rgb
R (8), G (8), B (8)
F, V, H,
Data Valid
Output Submodule:
If(rgb == red, reddone == 0)
redx = x;
redy = y;
reddone = 1;
If(rgb == green, greendone == 0)
greenx = x;
greeny = y;
greendone = 1;
If(x == 1024, y == 768)
ready = 1;
Else
ready = 0;
If(x == 0, y == 0)
reddone = 0;
greendone = 0;
Modified
NTSC to ZBT
X (10),
Y (10)
Data Valid
Pipeline
X (10),
Y (10)
Data Valid
Address (19),
Data (36),
WE
Redx, Redy, Greenx, Greeny,
Ready, Reddone, GreenDone
To 6.111 ZBT
To Game Logic
And Gesture Modules

Gesture Recognizer

Segment Encoding



0010 - Stop
0100 - Start

The segments are encoded such that the more similar the segments, the less the hamming distance between the encodings. This is used in our simple heuristic for computing score to take into account misinterpreted segments. Additionally special encodings indicate the start and stop of a gesture, that way if we get as input many segments fewer or more than the length of the gesture we can take this into account in our score.

Segment Determiner

Looks at the displacement between successive points, if they are far enough apart then count them, if they are not see if the gesture has ended. Every few points check the total movement and determine the segment, if we have enough of the same segment in a row, send this segment to the pattern determiner.

Pattern Determiner

Each gesture has its own module with a ROM storing the list of segments expected for that gesture and a register keeping track of the number of segments seen so far in the current gesture. Each time a new segment comes in we compare it to the expected segment (as well as previous and next segments in case one was dropped or added) and the score is updated. The maximizers update the current best pattern and if a stop segment is received we let the final score update and then compare it to a threshold. If a pattern matched sufficiently it is sent to the game logic.

Gesture Recognizer

X coord (10)

Y coord (10)

Ready

Segment Determiner

Registers:
- ~8 signed 11 bit registers for storing point data
- ~6 4-6 bit registers used for counting
- 1 ~20 bit counter used for speed

Computation:
- 2 signed 11 bit multipliers
- ~ 15 signed 11 bit adders/subtracters
- ~ 10 4-20 bit adders/subtracters
- ~ 4 right and left bit shifters

Time:
- ~ 10 clock cycles to analyze each coordinate pair

Scheduled Completion:
- In the process of testing initial verilog modules in ModelSim
- Expect to be done with testing by 11/17
- After that will modify as necessary to work with actual video data once that module is ready

Clock

Reset

4 Segment

Ready

3 Speed

Pattern Determiner

Patterns
Each pattern will have its own module that will take as input the segment and ready signals.

Registers:
- 1 5-bit reg that counts segments
- 4-bit x 32 ROM that stores the pattern
- 1 10-bit reg for score

Computation:
- ~ 5 10-bit adders
- ~5 10-bit shifters
- ~5 bit counters

Score 10

Pattern 6

Maximizers
A series of two 10-bit input maximizers will take the scores from the patterns and find the highest one.

Registers:
- 1 6-bit reg that stores pattern
- 1 10-bit reg for score

Computation:
- ~ 1 10-bit subtracter

Pattern 6

Speed 3

Ready

Time:
- ~ 8 clock cycles to analyze each segment

Scheduled Completion:
- In the process of testing initial verilog modules in ModelSim
- Expect to be done with testing by 11/17

Clock

Reset

The gesture recognizer analyzes the coordinates of the laser in real time, decomposing it into straight line segments. These segments are then analyzed in real time by modules representing each possible gesture, simple heuristics determine the score and are used to pick the matching pattern.

Game Logic

Input
- Red and Green coordinates, gesture and speed
- Keyboard input used to control players
- Vertical sync from the video output module

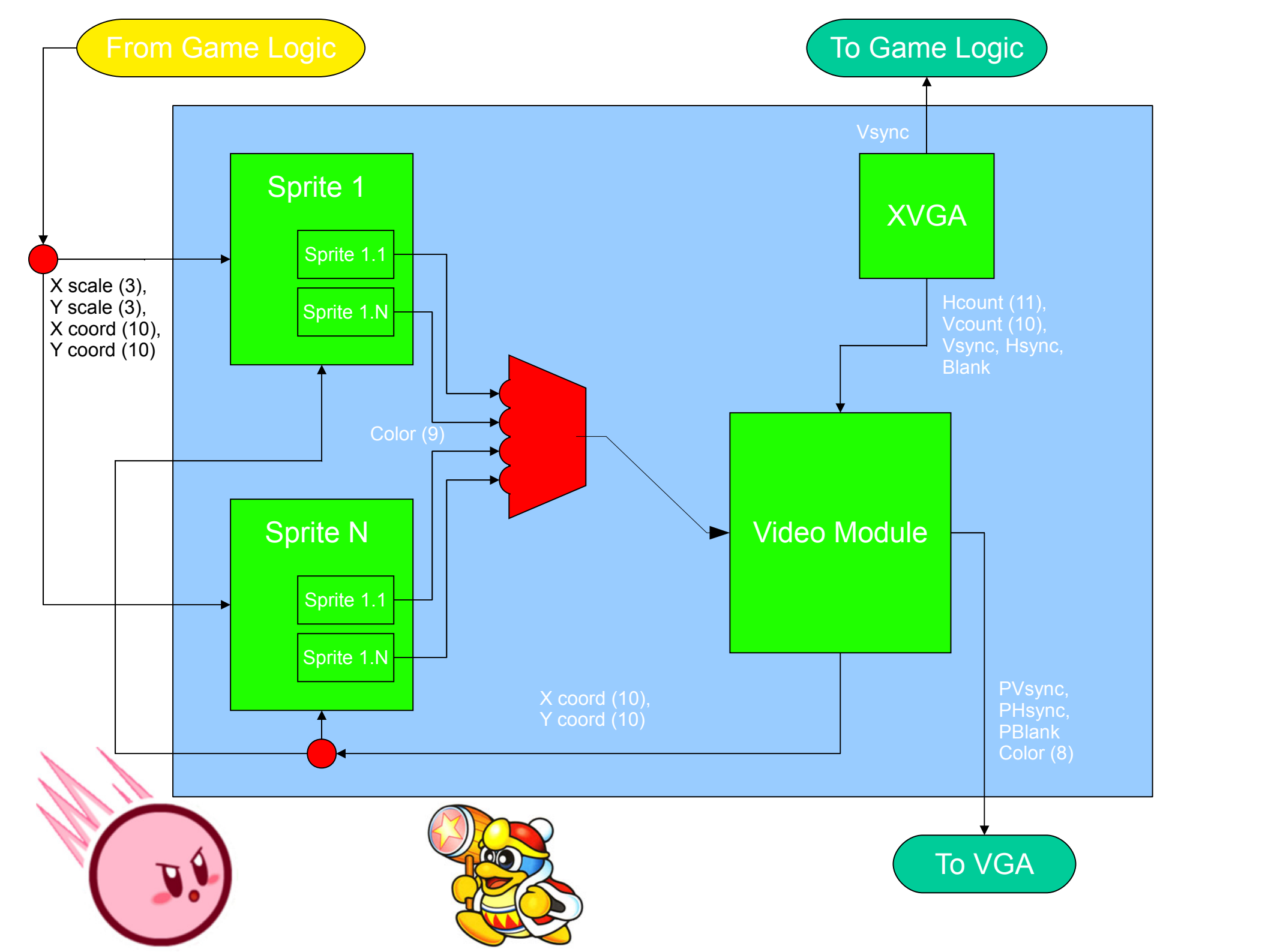
Output
- The position and size of every sprite

Registers
- Registers to keep track of the position and speed of every sprite
- Registers to keep track of player data (health, state (jumping etc.))

Computation
- Movement, and animation (changing sprites) of sprites – signed addition
- Collision detection for sprites – comparisons – signed addition, possibly multiplication for circles
- Simple physics (gravity, friction), counting etc. - signed and unsigned addition

Time
- Each new frame will be computed while the current one is being displayed with the new outputs stored in registers. When the vsync indicates that we are no longer drawing the current frame then the new outputs will be sent to all the sprites. We will have an entire frame to compute the next frame and sense the logic will be fairly simple and can be done in parallel we foresee no issues with completing the computations in time.

Scheduled Completion
- Upon completion simple functional versions of the video analyzer, and gesture recognition module, we will begin to work on this module. We will also update the related modules as necessary (add sprites, gestures etc.) until the game is really really awesome!


From Game Logic
To Game Logic
Vsync
XVGA
Sprite 1
Sprite 1.1
Sprite 1.N
X scale (3),
Y scale (3),
X coord (10),
Y coord (10)
Hcount (11),
Vcount (10),
Vsync, Hsync,
Blank
Color (9)
Video Module
Sprite N
Sprite 1.1
Sprite 1.N
X coord (10),
Y coord (10)
PVsync,
PHsync,
PBlank
Color (8)
To VGA

Schedule

Nov 9 – 15: Tynan: Complete testing of verilog gesture recognition modules, Dan: complete initial detection of one laser dot on white paper

Nov 16 – 22: Tynan: Implement gesture recognition with existing video system, Dan: complete video system to work with two dots on screen

Nov 23 – 29: Tynan: Finish deciding on overall game mechanics and begin game logic structure, Dan: Begin work on sprite and video output system

Nov 30 – Dec 6: Tynan: Finish game implementation, Dan: finish sprite implementation and all sprites

Dec 6 – Dec 8: Both: Finish writing up paper, tweak game as necessary and add awesome features, play game and celebrate being done