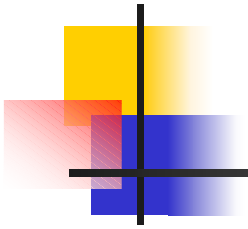
# iGamePlay

6.111 Final Project Presentation

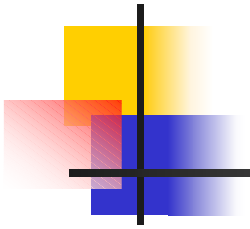
04/21/2004

By: Martijn Stevenson, Tom Wilson and Kale McNaney



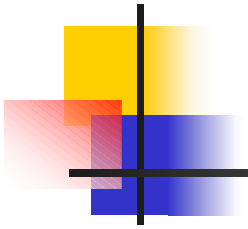
# Overview

- The big picture:
  - A game with two players (either cooperating or competing) with gameplay elements driven by musical cues.
- Two phases
  - Implementation/design: work out major technical issues. Progress: many issues already solved!
  - Content generation: actual game design phase, back-weighted. Progress: not so good.



# Major Modules

- Audio Processing
  - Generation of audio “cues” using spectrum analysis
- Game Logic/User Input
  - Capture user input and apply to game state
  - Interaction of users / sound-controlled game world
- Video Processing and Displays
  - Output game state to VGA using page buffering
  - Load sprites, backgrounds, etc. from ROM



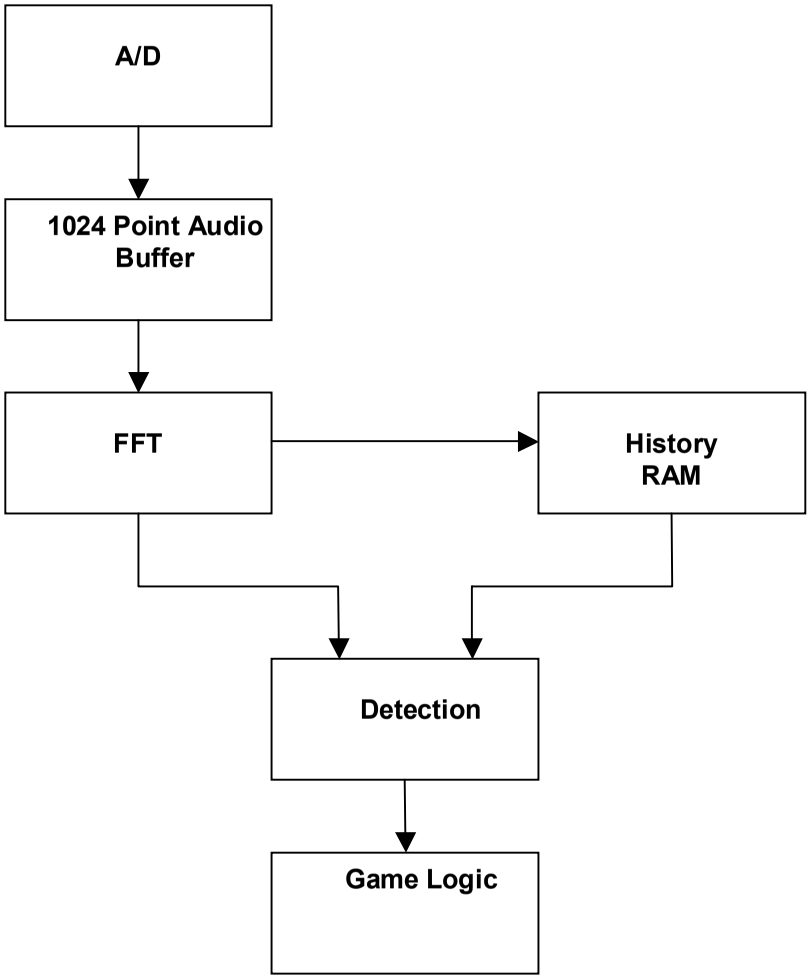
# Audio Overview

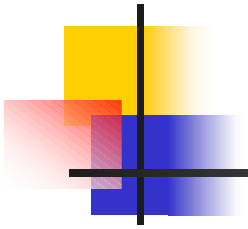
- iGamePlay uses arbitrary audio input to drive part of the game play
- Audio input is digitized for processing using AC'97 codec
  - Frames
    - Used to pass information from the controller (FPGA) to the Codec and vice versa
    - Comprised of 1 Tag @ 16 bits and12 slots @ 20 bits/slot – 256 bits total
    - New frame starts on low to high transition of SYNC signal
    - When controller sends a frame bit, codec simultaneously sends back a frame bit
  - Controller implemented using single FSM
  - Frames used to configure internal control registers
  - iGamePlay configured registers for Microphone line input to be digitized by ADC.
  - Digitized data, passed back in the frames from the codec, is used for processing

# Design For Cue Detection

- Codec Controller
  - Coded using single FSM
  - Runs asynchronously on 12.288 MHz BIT_CLK from codec
  - Codec can compute 1 new sample (left and right channel) every 20 us -- 50,000 samples computed every second
- Beat detection Algorithm
  - Gather 1024 samples from codec in a RAM
  - Use Xilinx 1024 point FFT Core module to get frequency representation of sample points
  - Store “instantaneous” frequency representation in a history buffer 48 addresses deep
  - Divide frequency representation into sub-bands
  - Compare “instantaneous” power in frequency to average power over the 48 in the history buffer
  - If comparison is above certain frequency, set cue bit high

# Audio Block Diagram

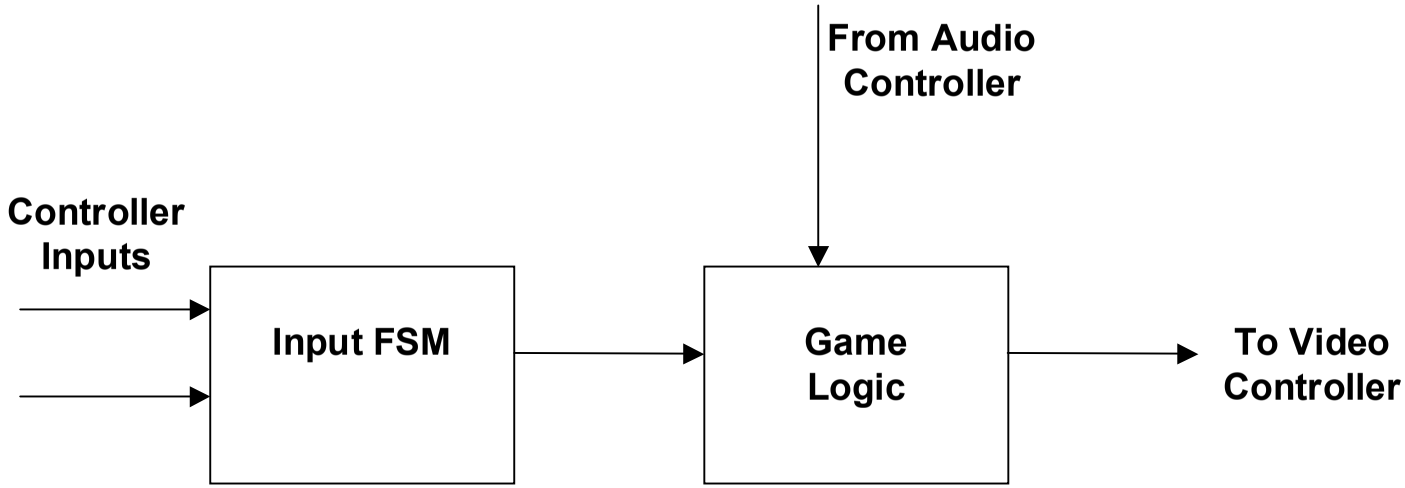




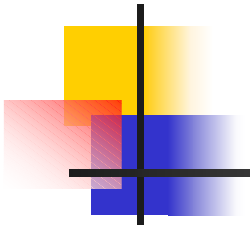
# Game Logic and User Inputs

- Challenges: programming, interactions between submodules
- Nintendo controllers – Input FSM
  - 3 signal wires + power + ground
  - @ 60 Hz, sample serial data stream – 8 pulses for 8 signals
- Game state – lots of communication between these
  - Game FSM
    - Player interactions: collisions, firing, game state
  - Player FSM
    - Player motion: debounce buttons, acceleration, friction
  - Missile FSM
    - Continue along direction fired, perhaps home in on target

# Game/Input Block Diagram

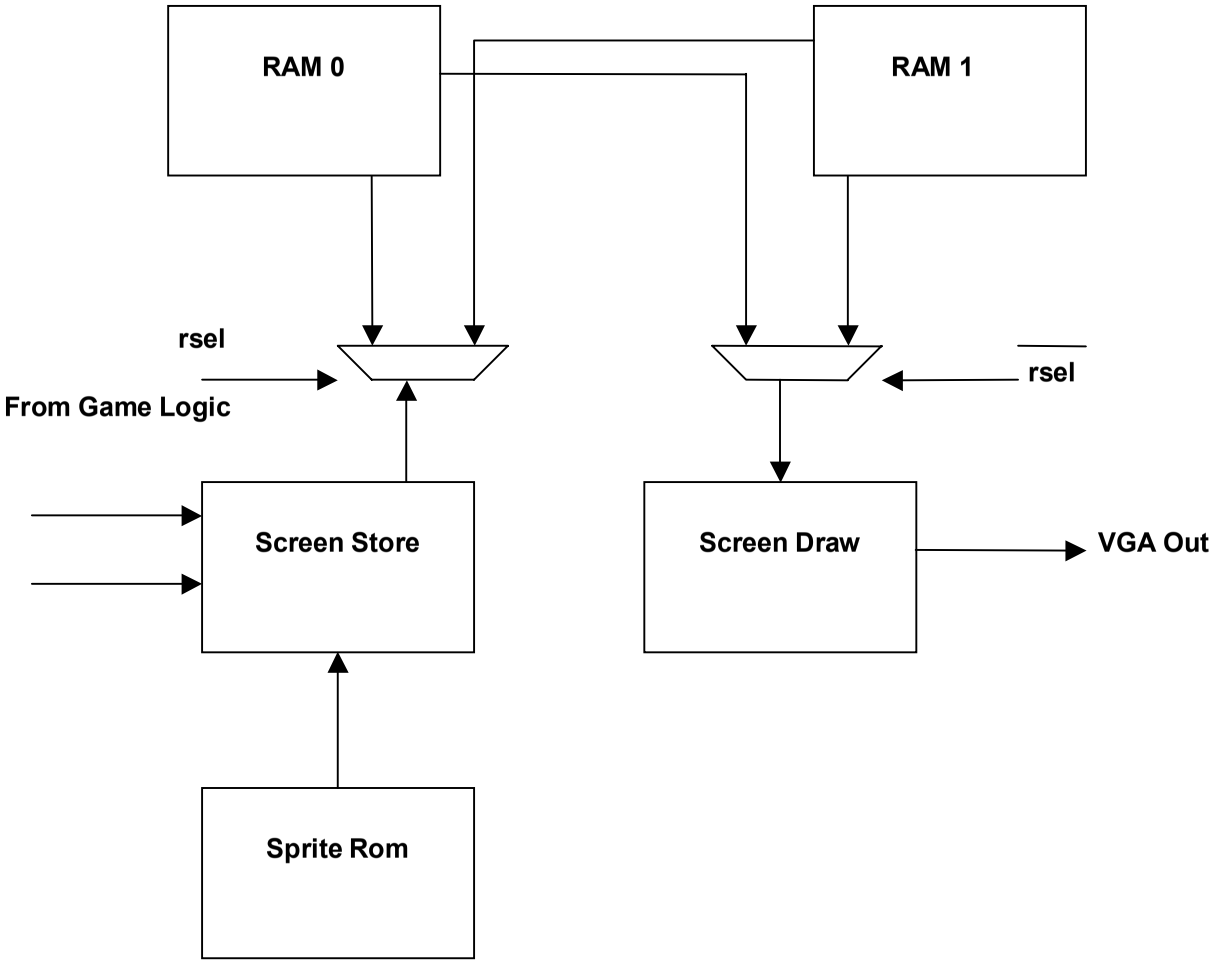


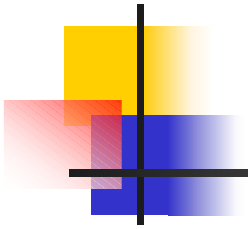
# Video Processing and Display



- ADV7185 chip
  - Control system generates timing signals (hsync, vsync, blanking)
  - Displays data from RAM
- Two ZBT RAMS
  - One RAM contains the current screen image
  - The other RAM stores the next screen
  - Control system swaps the RAMs every1/60$^{th}$ of a second
- System interface
  - Takes input from Game Logic system and sprite ROM
  - Stores video data to ZBT RAM

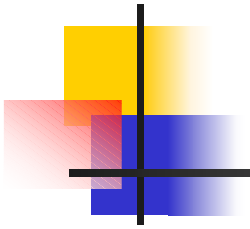
# Video Block Diagram





# Progress

- Configured AC'97 codec to sample analog input
- Started FFT of 1024 sample points
- Moved players using Nintendo controllers
- Implemented configurable screen wrapping, acceleration, friction, collision detection and (perhaps homing!) missiles
- Drew different colored squares to the screen
- Drew from ZBT RAM.
- Started page buffering



# Goals

- Determine 3 audio cues including Beat Detection
- Enemy or world movement to sound cues
- Menu, play, win modes
- Read video sprites from ROM
- Finish page buffering

Ideal world

- Arbitrary song support
- Sound effects with game
- Background morphs with sound