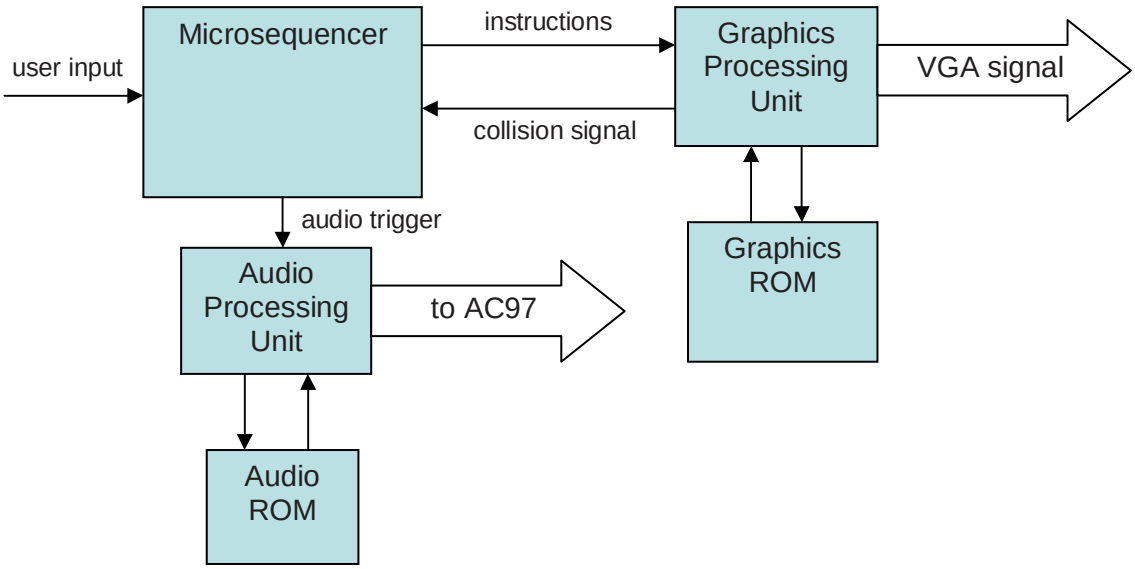
Project Proposal

Abstract
The project is similar to classic videogame side-scrollers, where players use direction keys and an action button to control a 2d character through plataforms, dodging or killing enemies and collecting items, using sprites and sprite collision, and a scrolling background. The game will progress through a series of stages, each with its time limit, to be completed by reaching its end. Sounds will be triggered, based on in-game events, such as collecting an item or dying, also with the possiblity for a continuous background music.

Implementation – high-level block diagram



The game logic, outside graphic processing, tends to be fairly simple; it moves around sprites in the screen, and triggers effects when collisions happen, such as a collision of the character with a tile representing the ground (stop the fall, kill the character depending on the type of tile he fell on), collision of the character with a tile representing an enemy (kill the character, or kill the enemy, or kill the projectile sprite and the enemy), collision of the character with an item or power-up (update score, update variables allowing new actions). As such, a microsequencer seems fit to deal with the game logic, as long as the graphics processing is moved to another logic block.

The microsequencer receives input from the user, as well as from sprite collisions. It will receive a signal from a slow (tens of Hz) from a hardware timer as well, as opposed to trying to emulate time counting on its program, to control movement of sprites and in-game time effects (such as a time limit).

The audio processing unit only receives control signals from the microsequencer, with ID of the audio to be played. It will keep playing a background music in loop, with up to 3 sounds being played simultaneously on its top. The sound effects will be stored in the

ROM as simple wave information; the background music might use sound synthesis, time allowing.

The graphics processing unit receives and returns information to the microsequencer. It is responsible for displaying a background panorama, game information (time remaining, lifes remaining, score), as well as the sprites for the game itself. Some “blocks” of impassable areas may be stored on local registers, to represent game entities such as the floor, or the ceiling; collisions of sprites with them are also notified, using a special reserved ID for representing a “blocked” area. The GPU is also responsible for managing up to S sprites simultaneously in the screen, coordinating which sprite ID given by the microsequencer is associated with which hardware sprite module. The microsequencer sends it informations such as creating, moving and destroying sprites, exchanging the background or panning the screen.

Each sprite has each pixel represented by X bits: R bits of red, G bits of green, B bits of blue, L bits of layer, a bit of opacity and a bit of collision. If two sprites take up the same pixel in the screen and both have the bit of collision set, a collision signal is emitted to the microsequencer. Additional logic is employed to send a collision signal only when the collision starts, as to not flood the microsequencer with those signals before the sprites can be moved.