EDA385 Bomberman

Fredrik Ahlberg
ael09fah@student.lu.se

Adam Johansson
rys08ajo@student.lu.se

Magnus Hultin
ael08mhu@student.lu.se

2013-09-10

1 Concept

Bomberman is a classic video game series, with the first incarnation published in 1983. The objective is to explore the map and take enemies out of action using bombs. The abilities of the player (i.e power and number of bombs) can be improved by power-ups.

This project aims to reimplement the multiplayer mode, where up to four players meet on a shared map.

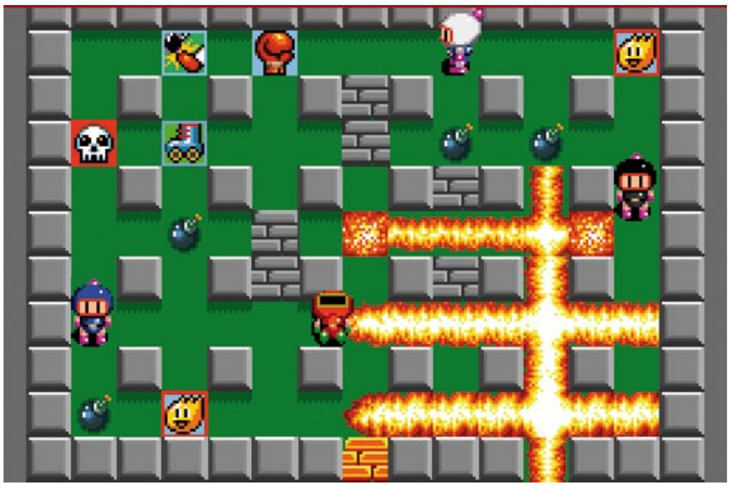

Figure 1: Screenshot from *Super Bomberman 2*. The image is cropped.

2 Implementation

2.1 System overview

The work of the console is coordinated by a *MicroBlaze* processor, which executes the game firmware from BRAM.

Attached to the system bus and thus the memory space of the processor is also:

- **A gamepad interface**.

  This should allow up to four game pads to be connected to the console and polled by the CPU.

- **A graphics controller**.

  This provides a memory and computionally efficient way of generating the complex graphics.

- **A sound generator**. *(Optional)*

  The game would not be as fun without sound effects and cheesy lo-fi music. This may be implemented as far as time allows. The structure of this block is to be determined.

The video controller generates a CPU interrupt at the end of each frame (*vblank*). The corresponding interrupt routine polls the gamepads, updates the game state and the sound playback, thus serving as the core of the game firmware.

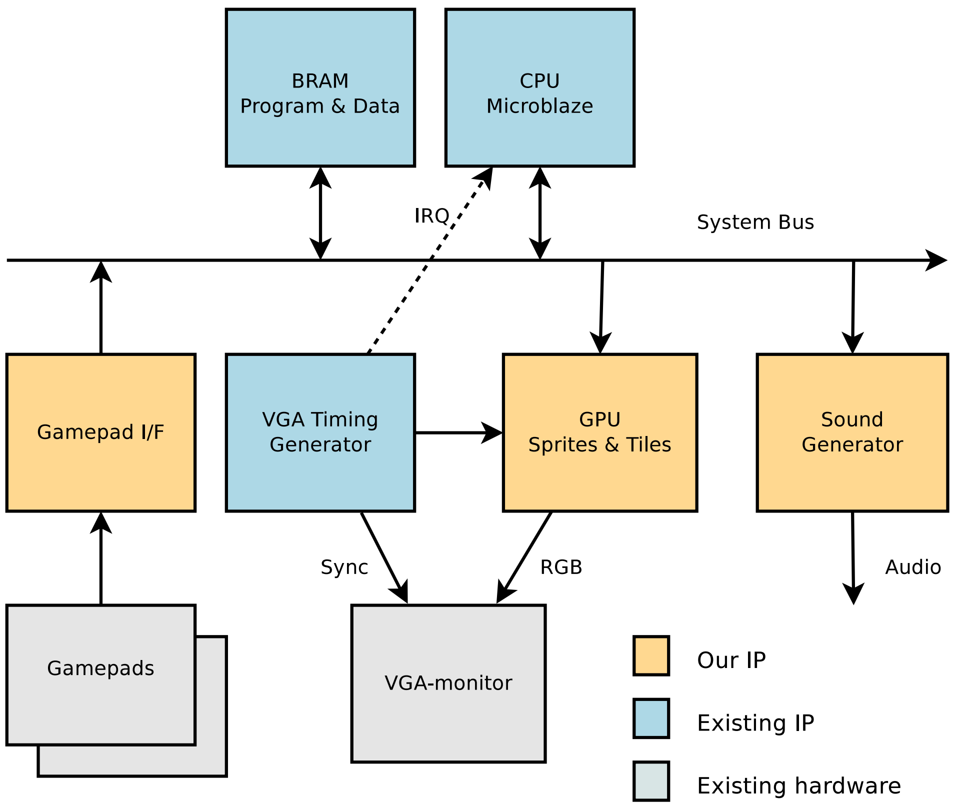

Figure 2: System block diagram

2.2 Graphics

Memory usage is a major constraint when dealing with graphics. To keep things simple, this subsystem is designed so as to avoid having to use external SDRAM and implementing a SDRAM controller, and instead putting the BRAM to good (and efficient) use.

The solution was a de facto standard in the video game industry; dividing the image into two or more planes, where static content like the background is described as a matrix of ”reusable” tiles (*tilemap*), and players and items are small, semitransparent images at arbitrary coordinates (*sprites*), drawn on top.

To further reduce memory usage, each pixel in the tiles and sprites is a actually *color index* in a palette, which is used to determine the color. A multitude of palettes can be stored in BRAM at the same time, so each entry in the tilemap or sprite attribute memory also specifies a *palette index*.

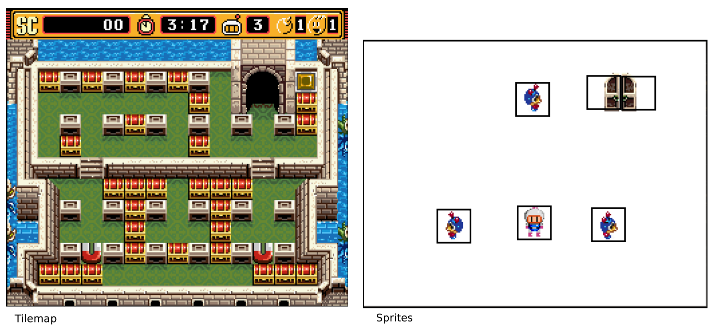

Figure 3: The tiled background and the sprites make up two separate graphic planes, which are then superimposed.

The tilemap and the sprites are rendered separately, and then muxed together right before the palette lookup. The background is rendered at pixel clock rate by cascading some BRAMs.

The sprite rendering is much more complex as a very large amount (512) of sprites may be defined at any time. The sprite plane is therefore rendered one scanline in advance and stored into a line buffer in BRAM.

A state machine clears the unused part of the line buffer, and then iterates the sprite attribute memory to find the sprites that intersect the currently rendered scanline. Those sprites are blitted into the line buffer, while considering the transparency of the sprites.

3 Schedule

The proposed schedule is shown below.

Software is developed in parallel with hardware using an emulator framework. A version control system (`git`) is used to allow safe and efficient parallel development of both software and hardware subsystems.

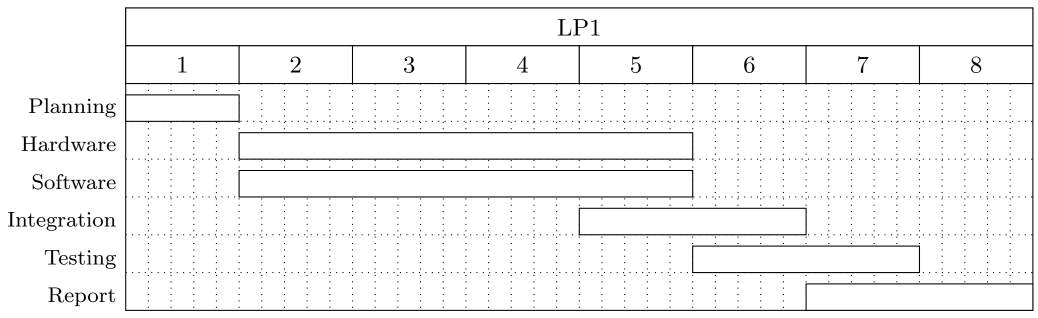

Figure 4: Proposed schedule